UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

FINDING MINIMUM GAPS AND DESIGNING KEY DERIVATION FUNCTIONS

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

DOCTOR OF PHILOSOPHY

By

YU-HSIN LI

Norman, Oklahoma

2011

FINDING MINIMUM GAPS AND DESIGNING KEY DERIVATION FUNCTIONS

A DISSERTATION APPROVED FOR THE
SCHOOL OF COMPUTER SCIENCE

BY

Dr. Qi Cheng, Chair

Dr. Sudarshan Dhall

Dr. Changwook Kim

Dr. Ralf Schmidt

Dr. Krishnaiyan Thulasiraman

To Mom and Dad

Acknowledgements

My first and foremost thanks and appreciation to Dr. Qi Cheng for persevering with me as my advisor throughout the time it took me to complete this research. He has provided guidance, support, ideas, and encouragement during my time at the University of Oklahoma. This dissertation would never be finished without him.

I appreciate my committee members, Dr. Sudarshan Dhall, Dr. Changwook Kim, Dr. Ralf Schmidt, and Dr. Krishnaiyan Thulasiraman, for their precious time and effort to read this dissertation and give their suggestions.

Many thanks to Jim Summers for his excellent server administration. This dissertation contains a large amount of computation on CS lab servers. Without sound working Linux machines, data generation would have difficulties.

I want to thank Barbara Bledsoe and Chyrl Yerdon for office support, for assisting with course enrollment, and for all sorts of tedious paper work. My special thanks to Barbara for her always kind, efficient, and helpful responses to me and everybody.

Last but definitely not the least, I would like to thank my wife, Chia-Hui Tsai, for her support and encouragement. I could not have completed this effort without her assistance, tolerance, and enthusiasm.

Contents

List of Tables

List of Figures

List of Algorithms

Abstract

The problem size gets larger as computers become faster. Using naive algorithms, even equipped with fast CPUs and large memories, computers still cannot handle many problems of certain size. Some searching tasks, however, can be answered with the help of the algorithmic technique, such as time and space trade-off.

Let k and n be positive integers, $n > k$. Define $r(n, k)$ to be the minimum positive value of

$$\left|\sqrt{a_1} + \cdots + \sqrt{a_k} - \sqrt{b_1} - \cdots - \sqrt{b_k}\right|$$

where $a_1, a_2, \cdots, a_k, b_1, b_2, \cdots, b_k$ are positive integers no larger than n. It is important to find a tight bound for $r(n, k)$, in connection to the sum-of-square-roots problem, a famous open problem in computational geometry. The current best lower bound and upper bound are far apart. For exact values of $r(n, k)$, only a few simple cases have been reported so far, and they can be found easily using exhaustive search. A new algorithm is developed to find $r(n, k)$ *exactly* in $n^{k+o(k)}$ time and in $n^{\lceil k/2 \rceil + o(k)}$ space. Space usage is decreased dramatically along with little increase in time, compared to an intuitive trade-off method. Our algorithm reduces time for swap-in and swap-out, minimizing the total running time. The problem is solved in size that was infeasible for a naive trade-off scheme. We also present lots of numerical data.

The time and space trade-off technique has its limitation. For some problems, when space is reduced to a certain extent, time will be increased exponentially. The trade-off technique does not apply to this situation. We explore such a property that discourages trade-off attacks.

Key generation is an important part of symmetric-key encryption algorithms, such

as AES. A key derivation function can be used to generate symmetric cipher session keys. As CPU technology advances, key derivation functions are more vulnerable to off-line brute force attacks. Based on the Memory Wall problem, we propose a simple number-theoretic way to mitigate exhaustive search attacks. We also present a formal definition of memory-bounded functions. On one hand, if attackers try to reduce memory usage, they are forced to spend dramatically more time. On the other hand, a memory-bound security scheme will minimize the difference between high-end and low-end computers. Trade-off attacks will hence be deterred.

Chapter 1

Introduction

Two important parts of a digital computer are central processing unit (CPU) and memory. The CPU reads data from memory, executes instructions, and stores results back to memory. Computers solve problems due to their CPU power and memory. The computation power is evaluated in cycles per unit time. A faster computer can run more cycles in the same amount of time and results in less CPU time. Memory as a storage space can be classified by accessing speed. Faster memory is costlier and hence smaller in capacity. With the advance of computer hardware technology, computers are capable of solving more complex problems than before.

To quote a Chinese proverb "While the priest climbs a post, the devil climbs ten". Problem size gets larger as computers become faster. Using naive algorithms, even equipped with fast CPUs and large memories, computers still cannot handle many problems of certain size. Some problems, however, can be answered with the help of algorithmic techniques.

Many exhaustive searching tasks, such as inversion of one-way functions and discrete logarithm problems, allow time-space trade-offs [Hellman80]. Time and space trade-off is an algorithmic technique to accelerate CPU throughput. Time refers to CPU time while space can be cache, memory, or hard drive. In this dissertation, space refers to main memory and will be used interchangeably. The key idea of trade-off schemes is reusability of one-time work: compute once and reuse again. From computational point of view, time can be saved by reusing pre-computed results stored in some space. As long as storage-lookup time is less than re-computation time, this

technique would be effective. Time and space trade-off is financially worthy. The cost to update a CPU is usually more expensive than to expand storage space. The advance of CPU technology is also faster than that of storage. The trade-off technique is a good choice when the budget to improve computer performance is limited.

1.1 Time and space trade-off

An example of time and space trade-off is to invert a one-way string permutation f, where sometimes exhaustive search is the only choice. Given a string y, we look for $f^{-1}(y)$ from N possible permutations such that $f(f^{-1}(y)) = y$. Let M be the total space and T be the total time required to derive an answer. We assume that storing a string needs $M = 1$ and that computing f once takes $T = 1$. One intuitive way, completely relying on CPU computations, costs $M = 1$ and $T = N$, resulting in minimum memory demand but slow speed. Another extreme way needs $M = N$ but is fast. It is composed of two phases, preprocessing (off-line) and on-line phase. The off-line phase stores all pairs of y and $f^{-1}(y)$ in a sorted table while the on-line phase answers by a lookup. The scheme depends absolutely on memory and costs $T = \log N$ and $M = N$. Although preprocessing time can be amortized by future searches, this extreme method consumes enormous space.

Hellman introduced a method to trade memory against time [Hellman80]. It is a middle ground between two extremes and is formed by preprocessing and on-line phases. Consider that each string is a point in a set of N permutations. The idea is to divide N by m, the number of chains, as shown in Figure 1.1. For each starting point, permute repeatedly t times to derive an endpoint. One endpoint takes t operations and hence m endpoints would cover all N points. To build a lookup table, the space requirement is $2m$, storing only pairs of starting and ending points. The preprocessing phase constructs such a table and sorts it by ending points. The time requirement

is still $tm = N$, apparently not a good method for a one-time-only inversion. The preprocessing time can be, however, amortized over the number of inversions, if the task needs to be conducted more than once. The on-line phase aims to match y in the sorted table. If y is not found, the user has to permute y and search for the new string; repeat till a match is found. When found, $f^{-1}(y)$ can be deduced by re-calculating the chain from the corresponding starting point. The on-line phase requires $T = t * log(m) = \frac{N}{m} * log(m)$ and $M = 2m$. The trade-off exists between T and M.

Naively applying the technique will not benefit us much. Space requirement would still be huge for certain problems, such as finding the smallest gap between sums of square roots. The plain trade-off scheme works for certain problem sizes if space demands are feasible. For larger sizes, a better technique is expected.

1.2 The smallest gap between sums of square roots

Comparing the lengths of two polygonal paths can be treated as calculating the difference of the sums of square roots, as nodes are on integral coordinates in a two-

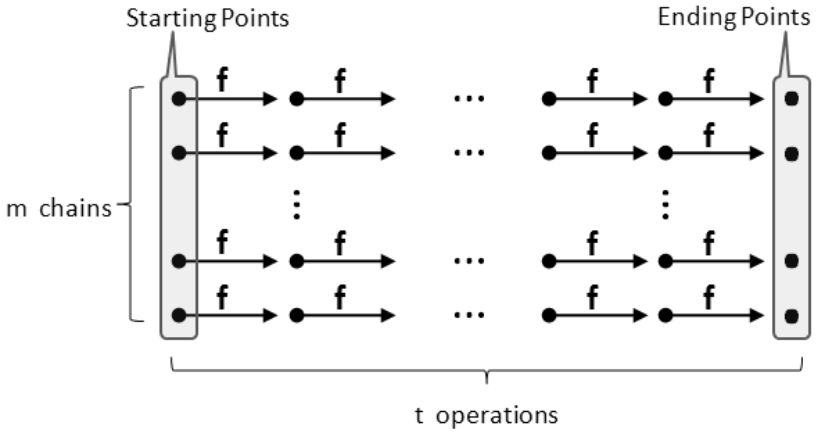

Figure 1.1: Hellman's cryptanalytic time-space trade-off

dimensional plane. The minimum nonzero difference has been an open problem for decades.

An efficient algorithm is developed to search for the minimum difference between the sums of square roots of small integers. As a typical searching task, the time and space trade-off technique is employed to save computation time. Our technique further improves the naive trade-off scheme. Space usage is decreased dramatically along with little increase in time, compared to an intuitive way. Our technique reduces time for swap-in and swap-out, minimizing the total running time. The problem is solved for sizes infeasible for the naive trade-off scheme.

1.3 Memory-bounded moderately hard functions

Slow computation, as an access-control mechanism, is preferred in some situations. It hinders a large surge of using certain computer resources, e.g. impediment to denial of service (DoS) attacks. DoS attack floods a network service with requests in a short time preventing legitimate users from the service. Examples of attacks include TCP SYN flooding and HTTP request flooding [JKR02]. The idea is to take advantage of fast and free computing resources. Most network services rely on these characteristics

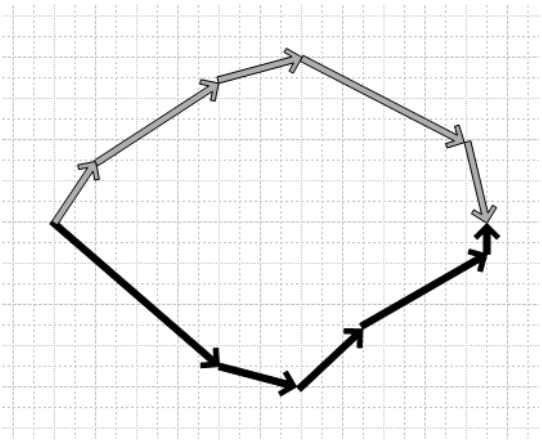

Figure 1.2: The minimum nonzero difference between two paths

to attract customers. Malicious users exploit properties to obstruct traffic.

Some problems can be solved in polynomial time. They are considered easy problems. There exist situations when space is reduced to certain extent, time will be increased exponentially. Easy problems become hard ones. The trade-off technique does not apply to them. We want to explore such property that discourages trade-off attacks.

The anti-tradeoff property provides a solution against DoS attacks. Forcing to walk randomly in certain amount of memory is a good access-control mechanism. On one hand, it will be hard to attack. Trying to reduce memory amount usage, malicious users result in dramatically paying more time instead. This is a direction for security schemes. On the other hand, memory-bound control schemes will minimize differences between high-end and low-end computers.

1.4 Results and structure

The sum-of-square-roots problem is a famous open problem in computational geometry. The best lower bound and upper bound are far apart. Chapter 2 introduces a new upper bound. An algorithm in Chapter 3 finds exactly the minimum gap for small integers. The result has been published in the proceeding of the 9th Latin American Theoretical Informatics Symposium (LATIN 2010) [CL10].

Memory-bounded functions have been designed to combat email spams in a sequence of papers [ABMW03, DGN03, DNW05] but lacked for a formal definition. The first formal definition is given in Chapter 4. A number-theoretic construction is in Chapter 5. The result has been published in the International Symposium on Trusted Computing [CL08]. Conclusion and future work will be discussed in Chapter 6.

Chapter 2

Sums of Square Roots

2.1 Introduction

In computational geometry, a fundamental problem is to compare lengths of two polygonal paths. On a two-dimensional plane, if points have integer coordinates, the length of a line can be expressed in a square root of an integer because of the Pythagorean Theorem. Several connected such lines form a polygonal path. The length of a path can be expressed in the sum of square roots of integer. Given two paths, we want to know their difference to determine a shorter one.

The problem of comparing two sums of square roots of integer exists in Turing Machine model. Computational geometry relies on computers. Computers nowadays are an implementation of the Turing machine model. Compared to the real-number model, Turing machine has its own limitation to handle this problem. In computational geometry one sometimes assumes a model of real-number machines, where one memory cell can hold one real number. It is then assumed that an algebraic operation, taking a square root as well as a comparison between real numbers can be done in one operation. There is a straight forward way to compare sums of square roots in a real-number machine. But this model is not realistic, as shown in [Shamir79, Schonhage79].

The geometrical question can be reduced to a numerical problem of comparing two sums of square roots of integers.

Definition 2.1.1. For positive integers n, k, a_i, b_i. Let

$$r(n,k) \stackrel{\text{def}}{=} \min_{1 \le a_i, b_i \le n} \left(\left| \sum_{i=0}^{k-1} \sqrt{a_i} - \sum_{i=0}^{k-1} \sqrt{b_i} \right| \right) \neq 0.$$

$r(n,k)$ describes the smallest gap between two sums while $-\log r(n,k)$ represents the number of digits of precision needed. We try to find an upper bound of $-\log r(n,k)$ as a function of n and k. This has been an open problem for decades. The origin of it can be dated back to 1981 [O'Rourke81], or even earlier. So far, only weak bounds have been found [DMO01].

2.2 Related work

Angluin and Eisenstat [AE04] considered the case $k = 2$ and gave bounds on the minimum nonzero separation of the sum of two square roots of positive integers from an integer. They proved that $r(n,2) = \Theta(\frac{1}{n^{3/2}})$.

Using the root separation method, Burnikel et al. [BFMS00] proved that $-\log r(n,k) = O(2^{2k} \log n)$.

The Prouhet–Tarry–Escott problem asks for two disjoint sets A and B of n integers each, such that: $\sum_{a \in A} a^i = \sum_{b \in B} b^i$ for each integer i from 1 to a given k. For a fixed k, $\Omega(k \log n - \frac{1}{2} \log n)$ is a lower bound of $-\log r(n,k)$ due to Ronald Graham [QW06].

Qian and Wang [QW06] gave a constructive upper bound of $r(n,k) = O(n^{-2k+\frac{3}{2}})$, which is a lower bound of $-\log r(n,k) = \Omega(k \log n)$. They also conjectured that $\log r(n,k) = \Theta(n^{\frac{1}{2}-2^{k-2}})$.

Cheng [Cheng06] gave an upper bound $-\log r(n,k) = 2^{O(n/\log n)}$ which beats the root separation bound as long as $n \le ck \log k$ for some constant c.

There is a wide gap between the known upper bound and the lower bound of

$-\log r(n,k)$. Until the fundamental problem has been resolved, we cannot even put the presumably easy problem such as Euclidean Minimum Spanning Tree problem in **P** (the polynomial-time class), and the Euclidean Traveling Salesman problem in **NP** (the nondeterministic polynomial-time class).

2.3 An upper bound of the smallest gap

Qian-Wang's upper bound was derived from the inequality:

$$0 < \left| \sum_{i=0}^{2k-1} \binom{2k-1}{i} (-1)^i \sqrt{t+i} \right| \leq \frac{1 \times 3 \times 5 \times \cdots \times (4k-5)}{2^{2k-1} t^{2k-\frac{3}{2}}}.$$

Let $a_i = \binom{2k-1}{2i-2}^2 (t+2i-2)$ for $1 \leq i \leq k$ and $b_i = \binom{2k-1}{2i-1}^2 (t+2i-1)$ for $1 \leq i \leq k$. Then we have

$$0 < \left| \sum_{i=1}^{k} \sqrt{a_i} - \sum_{i=1}^{k} \sqrt{b_i} \right| \leq \frac{1 \times 3 \times 5 \times \cdots \times (4k-5)}{2^{2k-1} t^{2k-\frac{3}{2}}}.$$

Note that $\binom{2k-1}{i}$ can be as large as $\binom{2k-1}{k} \geq 2^{2k-1}/(2k)$. To get an upper bound for $r(n,k)$, assign

$$n = \binom{2k-1}{k}^2 (t+k),$$

thus we have $-\log r(n,k) \geq 2k \log n - 8k^2 + O(\log n + k \log k)$. Hence Qian and Wang's result only applies when n is much greater than 2^{4k}. In particular it does not give a meaningful bound for small n and small k, for instance, $r(100,7)$.

Another interesting upper bound depends on the Prouhet–Tarry–Escott problem, which is to find a solution for a system of equations:

$$\sum_{i=1}^{k} a_i^t = \sum_{i=1}^{k} b_i^t \ , \quad 1 \leq t \leq k-1$$

under the condition that $a_1 \leq a_2 \cdots \leq a_k$ and $b_1 \leq b_2 \cdots \leq b_k$ are distinct lists

of integers. No such solutions have, however, been found for $k = 11$ and $k > 13$ [BLP03]. Therefore the approach based on the Prouhet–Tarry–Escott problem is not scalable.

Here we present an upper bound based on the pigeonhole argument.

Definition 2.3.1. An integer n is **square-free** if there is no integer $a > 1$ such that $a^2 \Big| n$. We use $s(n)$ to denote the number of positive square-free integers less than n, e.g. $s(100) = 61$.

Proposition 2.3.2. *The set* $\{\sqrt{n} \Big| n \in \mathbb{N}$ *is square-free*$\}$ *is linearly independent over rationals.*

Proof. See A.0.4. □

Theorem 2.3.3. *We have*

$$r(n,k) \leq \frac{k\sqrt{n} - k}{\binom{s(n)+k-1}{k} - 1}.$$

Proof. Consider the set $\left\{(a_1, a_2, \cdots, a_k) \Big| a_i \text{ is squaer-free}, 1 \leq a_1 \leq a_2 \leq \cdots \leq a_k \leq n.\right\}$ The set has cardinality $\binom{s(n)+k-1}{k}$. For each element $(a_1, a_2, \cdots, a_k)$ in the set, the sum $\sum_{i=1}^{k} \sqrt{a_i}$ is distinct by Proposition 2.3.2. Hence there are $\binom{s(n)+k-1}{k}$ many distinct sums in the range $[k, k\sqrt{n}]$. There must be two points within the distance $\frac{k\sqrt{n}-k}{\binom{s(n)+k-1}{k}-1}$ from each other. The theorem follows. □

From this, one can derive

Corollary 2.3.4. $-\log r(n,k) \geq k \log n - k \log k + O(\log(nk))$

Note that in comparison to the Qian–Wang's bound, this is weaker when n is very large. But it is better when n is a polynomial in k. Hence, it has wider applicability. For example, when $n = 100$ and $k = 7$, it can give us a meaningful upper bound: $r(100, 7) \leq 7.2 \times 10^{-8}$.

Chapter 3

The Smallest Gap Between Sums of Square Roots of Small Integers

3.1 Motivation

How close is the bound to reality? We need a provable bound which represent the actual situation. The bound should be as tight as possible. A way to show this is by running some feasible cases. In addition, numerical data shed lights on the type of integers whose square roots summations are extremely close.

So far only a few toy examples have been reported and they can be found easily using an exhaustive search:

$$r(20, 2) \approx .0002 = \sqrt{10} + \sqrt{11} - \sqrt{5} - \sqrt{18}.$$

$$r(20, 3) \approx .000005 = \sqrt{5} + \sqrt{6} + \sqrt{18} - \sqrt{4} - \sqrt{12} - \sqrt{12}.$$

Computing power has gradually increased every year which allows us to go beyond toy examples. Nevertheless, it still has its limitation. Our extensive numerical studies cover only small n and k.

In many practical situations, especially in the exact geometric computation, n and k are small. Explicit bounds like one we produce here help to decide how much precision is needed.

Moreover, since the upper bound is so far away from the lower bound, the numerical data may provide us some hints on which bound is closer to the truth and may

inspire us to formulate a reasonable conjecture on a tight bound of $r(n, k)$.

3.2 Space efficient technique

We are looking for the smallest difference between any two sums of square roots of integers by running cases on small n and k. To the best of our knowledge, there is no better way other than exhaustive search.

The smallest non-zero difference only is due to two consecutive sums. Let S be a set of sums of all combinations in form $\sqrt{a_1}+\sqrt{a_2}+\sqrt{a_3}+\cdots+\sqrt{a_k}$, where $1 \leq a_i \leq n$. For a given sum $s_i \in S$, the smallest difference of $|s_i - s_j|$ occurs when $|i - j| = 1$, where i, j are indices of sorted S. Otherwise, there will always be a smaller one. Hence the first task is to sort the sums of all combinations.

In terms of combinations, we need to enumerate all combinations with repetition. For example, for $n = 2$ and $k = 3$, we have $\{\sqrt{1} + \sqrt{1} + \sqrt{1}, \sqrt{1} + \sqrt{1} + \sqrt{2}, \sqrt{1} + \sqrt{2} + \sqrt{2}, \sqrt{2} + \sqrt{2} + \sqrt{2}\}$. According to Euler, the number of k-combinations, with repetitions, from n distinct object is

$$|(n, k)| = \frac{(n + k - 1)!}{k!(n - 1)!} = \binom{n + k - 1}{k}.$$

The storage of all combinations takes up lots of memory space. A primitive double data type, which takes up 8 bytes, does not have enough precision. Instead, we use double-double [QD08], which has approximately 32 decimal digits and requires 16 bytes. A middle-to-high end computer nowadays has memory about 8 gigabytes. To handle the case $(n, k) = (100, 7)$, we need 45.4 gigabytes of memory space. It is prohibitive to handle such cases without a space-efficient algorithm.

Number theorists have been using a space-saving mechanism to test difficult conjectures on computers. For example, consider the following diophantine equation: $a^4 + b^4 + c^4 = d^4$. Bernstein's idea [Bernstein01] was to build two streams of sorted

integers, one for $a^4 + b^4$ and another one for $d^4 - c^4$, and then look for collisions. Algorithm 1 presents the mechanism. With this idea, the space requirement can be saved by $|S|^{\frac{1}{2}}$.

Algorithm 3.1: The space-saving mechanism for enumeration

Input: P is a sorted list of $\left|(n, k - \lceil \frac{k}{2} \rceil)\right|$ elements.
Q is a sorted list of $\left|(n, k - \lfloor \frac{k}{2} \rfloor)\right|$ elements.
Output: The sorted list of $|(n, k)|$ elements.

```
Build a heap for P[i]||Q[1], 1 ≤ i ≤ sizeof(P);
while sizeof(P) ≠ 0 do
    Pop the root element, P[i]||Q[j], from heap;
    if j < sizeof(Q) then
        Push P[i]||Q[j + 1] into heap;
    end
    Re-heap;
end
```

The use of heaps to enumerate the sums in a sorted order appeared quite early [Knuth73, Section 5.2.3]. Let P and Q be sorted lists and let $P[i], Q[i]$ be the their ith element, respectively. Denote element concatenation by $\|$. The above algorithm dynamically enumerates one element per iteration in sorted order, avoiding massive storage requirement. This is an important technique to perform exhaustive search beyond toy examples.

3.3 Algorithm for finding the gap

We present Algorithm 2 to compute $r(n, k)$ exactly based on the idea of enumerating summations using a heap. It further improves the execution performance: instead of $P[i]\|Q[j + 1]$, Algorithm 2 pushes $P[i]\|Q[j']$ into the heap.

Theorem 3.3.1. *When Algorithm 2 halts, it outputs* $r(n, k)$.

Algorithm 3.2: Algorithm for finding $r(n,k)$

Input: Two positive integers n, k $(n > k)$.

Output: $r(n,k) =$ smallestDifference.

Let P be an array containing all k-combinations, with repetitions, from 1 to n. Let $(a_1, a_2, \ldots, a_A)$ be an element in P, where $1 \leq a_1 \leq a_2 \leq \cdots \leq a_A \leq n$ and $A = \lfloor \frac{k}{2} \rfloor$;

Sort P according to $\sum_{i=1}^{A} \sqrt{a_i}$;

Let Q be an array containing all $(k-A)$-combinations, with repetitions, from 1 to n. Let $(a_1, a_2, \ldots, a_{k-A})$ be an element in Q, where $1 \leq a_1 \leq a_2 \leq \cdots \leq a_{k-A} \leq n$;

Sort Q according to $\sum_{i=1}^{k-A} \sqrt{a_i}$;

smallestDifference $\leftarrow \infty$;

previousRootValue $\leftarrow 0$;

Denote element concatenation by $\|$. Build a heap for $P[i]\|Q[1]$ according to $\sum_{l=1}^{A} \sqrt{P[i][l]} + \sum_{l=1}^{k-A} \sqrt{Q[j][l]}$, where $1 \leq i \leq sizeof(P)$ and $1 \leq j \leq sizeof(Q)$.;

while $sizeof(P) \neq 0$ **do**

    Pop the root element, $P[i]\|Q[j]$, from heap;

    currentRootValue $\leftarrow \sum_{l=1}^{A} \sqrt{P[i][l]} + \sum_{l=1}^{k-A} \sqrt{Q[j][l]}$;

    if $0 <$ |currentRootValue − previousRootValue| $<$ smallestDifference **then**

        smallestDifference $\leftarrow$ |currentRootValue − previousRootValue|;

    end

    previousRootValue $\leftarrow$ currentRootValue;

    if $\exists j < j' < sizeof(Q)$ such that $P[i][A] \leq Q[j'][1]$ **then**

        Push $P[i]\|Q[j']$ into heap;

    end

    Re-heap;

end

Proof. For any $1 \leq a_1 \leq a_2 \cdots \leq a_A \leq n$, define

$$S_{a_1,a_2,\cdots,a_A} = \{(a_1, a_2, \cdots, a_k) \Big| a_A \leq a_{A+1} \leq a_{A+2} \leq \cdots \leq a_k \leq n\}$$

Partition the set $S = \{(a_1, a_2, \cdots, a_k) \Big| 1 \leq a_1 \leq a_2 \leq \cdots \leq a_k \leq n\}$ into subsets according to the first A elements, namely,

$$S = \bigcup_{1 \leq a_1 \leq a_2 \leq \cdots \leq a_A \leq n} S_{a_1,a_2,\cdots,a_A}.$$

As usual, we order two lists of integers by their sums of square roots. Consider the following procedure: select the smallest element among all the the minimum elements in all the subsets, and remove it from the subset. If we repeat the procedure, we generate a stream of elements of S in a sorted order.

It can be verified that in our algorithm, the heap consists of exactly all the minimum elements from all the subsets. The root of the heap contains the minimum element of the heap. After we remove the element at the root, we put the next element from its subset into the heap. Hence the algorithm produces a stream of elements from S in a sorted order. The minimum gap between two consecutive elements in the stream is $r(n, k)$ by definition. □

Our search reveals that $r(100, 8) = 2.77 \times 10^{-21}$, which is reached by $\sqrt{16}+\sqrt{43}+\sqrt{43}+\sqrt{46}+\sqrt{60}+\sqrt{85}+\sqrt{89}+\sqrt{95}$, which is 60.043493658302558242272654 98 and $\sqrt{7}+\sqrt{41}+\sqrt{42}+\sqrt{51}+\sqrt{76}+\sqrt{83}+\sqrt{94}+\sqrt{97}$, which is 60.043493658302558242269883 31.

3.4 Time and space complexity

Without using Algorithm 1, a naive exhaustive search algorithm needs $O(n^k)$ space. Our means aims at the smallest difference. Once a new element is enumerated, the gap will be recorded, then the element can be freed.

Our algorithm uses much lesser space than the sorting approach while preserving the time complexity, which makes computing $r(100, 8)$ feasible.

Theorem 3.4.1. *The algorithm runs in time at most $n^{k+o(k)}$ and space at most $n^{\lceil \frac{k}{2} \rceil + o(k)}$.*

Proof. Using the root separation bound, we need at most $O(2^{2k} \log n)$ bit to represent the sum of square roots for comparison purposes. So comparing two elements takes time $(2^{2k} \log n)^{O(1)}$. Since every element in S appears at the root of the heap at most once and $|S| \leq n^k$, the main loop has at most n^k iterations. For each iteration, the time complexity is

$$(2^{2k} \log n)^{O(1)} \log(n^{\lceil \frac{k}{2} \rceil}).$$

The complexity of other steps are much smaller comparing to the loop. Hence the time complexity is $n^{k+o(k)}$. The space complexity is clearly $n^{\lceil \frac{k}{2} \rceil + o(k)}$. □

3.5 Numerical data and observations

To implement our algorithm, the main issue is to decide the precision when computing the square roots and their summations. We need to pay attention to two possibilities: First, two summations may be different, but if the precision is set too small, then they appear to be equal numerically. Keep in mind that we have not ruled out that $r(n, k)$ can be as small as n^{-2^k}. Secondly two expressions may represent the same real number, but after the numerical calculation, they are different. This is the issue of numerical stability. In either case, we may get a wrong $r(n, k)$.

Our strategy is to set the precision at about $2k \log n$ decimal digits. For example, to compute $r(100, 8)$, we use the data type which has precision about 28 decimal digits. Whenever the difference of two summations is smaller than n^{-2k}, we call a procedure based on Proposition 2.3.2 to decide whether the two numbers are equal or not.

We produce some statistical data about the sums of square roots and the gaps between two consecutive sums. On the same high-end PC, the computation takes about 18 hours to find $r(100, 7)$ and about 30 days for $r(100, 8)$. There are $217,538,310,639$ numbers in $[8, 100]$ which can be written as summations of 8 square roots of positive integers less than 100. Hence there are $217,538,310,638$ gaps between two consecutive numbers after we sort all the sums.

In Table B.6 and Figure B.6, we list an integer $8 \leq a \leq 80$ with the number of α such that $\lfloor \alpha \rfloor = a$ and α can be represented as $\sqrt{a_1} + \sqrt{a_2} + \cdots + \sqrt{a_8}$ ($1 \leq a_1 \leq a_2 \cdots \leq a_8 \leq 100$). Note that if two summations have the same value, they are counted only once. From the table, we see that there are $13,281,868,775$ sums in the $[55, 56)$, which gives us a more precise pigeonhole upper bound for $r(n, k)$ at $1/13281868775 = 7.529 \times 10^{-11}$, which is still several magnitudes away from $r(n, k)$.

In Table C.6 and Figure C.6, for each range, we list the number of gaps between consecutive numbers in the range. From the table, we see that there are 4 gaps which have the maximum magnitude at 10^{-21}.

Table 3.1: The smallest difference of (n, k)

n	**k**	$\mathbf{r(n,k)}$	**Sum of Square Roots**
100000	2	6.58×10^{-18}	$(\sqrt{47035} + \sqrt{82802})$ - $(\sqrt{43728} + \sqrt{87330})$
100	2	1.53×10^{-07}	$(\sqrt{33} + \sqrt{74})$ - $(\sqrt{28} + \sqrt{82})$
5000	3	2.84×10^{-20}	$(\sqrt{29} + \sqrt{1097} + \sqrt{3153})$ - $(\sqrt{226} + \sqrt{987} + \sqrt{2324})$
100	3	8.45×10^{-10}	$(\sqrt{31} + \sqrt{48} + \sqrt{98})$ - $(\sqrt{42} + \sqrt{42} + \sqrt{89})$
1000	4	9.15×10^{-20}	$(\sqrt{154} + \sqrt{381} + \sqrt{770} + \sqrt{774})$ - $(\sqrt{128} + \sqrt{394} + \sqrt{637} + \sqrt{967})$
100	4	5.04×10^{-14}	$(\sqrt{45} + \sqrt{63} + \sqrt{91} + \sqrt{96})$ - $(\sqrt{44} + \sqrt{65} + \sqrt{93} + \sqrt{93})$
300	5	1.45×10^{-19}	$(\sqrt{101} + \sqrt{131} + \sqrt{185} + \sqrt{211} + \sqrt{212})$ - $(\sqrt{61} + \sqrt{128} + \sqrt{154} + \sqrt{264} + \sqrt{269})$
100	5	5.66×10^{-15}	$(\sqrt{36} + \sqrt{40} + \sqrt{83} + \sqrt{86} + \sqrt{94})$ - $(\sqrt{52} + \sqrt{62} + \sqrt{66} + \sqrt{69} + \sqrt{79})$
150	6	3.97×10^{-19}	$(\sqrt{34} + \sqrt{36} + \sqrt{57} + \sqrt{76} + \sqrt{92} + \sqrt{149})$ - $(\sqrt{11} + \sqrt{35} + \sqrt{52} + \sqrt{95} + \sqrt{139} + \sqrt{142})$
100	6	2.89×10^{-17}	$(\sqrt{21} + \sqrt{54} + \sqrt{62} + \sqrt{67} + \sqrt{92} + \sqrt{99})$ - $(\sqrt{15} + \sqrt{59} + \sqrt{76} + \sqrt{76} + \sqrt{82} + \sqrt{90})$
100	7	1.88×10^{-19}	$\sqrt{7} + \sqrt{14} + \sqrt{39} + \sqrt{70} + \sqrt{72} + \sqrt{76} + \sqrt{85}$ $\sqrt{13} + \sqrt{16} + \sqrt{46} + \sqrt{55} + \sqrt{67} + \sqrt{73} + \sqrt{79}$
100	8	2.77×10^{-21}	$(\sqrt{16} + \sqrt{43} + \sqrt{43} + \sqrt{46} + \sqrt{60} + \sqrt{85} + \sqrt{89} + \sqrt{95})$ - $(\sqrt{7} + \sqrt{41} + \sqrt{42} + \sqrt{51} + \sqrt{76} + \sqrt{83} + \sqrt{94} + \sqrt{97})$

Chapter 4

Moderately Hard Functions

4.1 Motivation

Functions are mappings from strings to strings. One function can be computed by different algorithms. A function is considered hard if there is no known efficient algorithm to compute it. Easy functions are suitable for completing tasks efficiently while hard ones deter abusers. Cryptographic schemes are built upon these hardness to ensure security. We want, for instance, that finding the inverses of one-way functions is infeasible on any modern computer.

There are situations to apply neither too hard nor too easy functions. Moderately hard functions are suitable and are useful for access control. If it is too easy for a user to send any amount of email to many users, spams will be a problem. A resource is usually shared by different users. To regulate its usage, a user is required to make some extra efforts. That is, the machine needs to spend more time and/or space.

The idea is first introduced by Cynthia Dwork and Moni Naor [DN92]. They call this kind of function a pricing function, as a user needs to pay a fee, in terms of computing resources, in order to use the service. This is a means to discourage abuse of resource. In general, a moderately hard function can be used to implement an access control scheme.

In theoretical computer science and cryptography, a function is generally considered easy if the complexity of its algorithm is in **P**. Not all polynomial-time algorithms are, however, fast in reality. Users may feel prominent difference in running

algorithms with complexity in $O(n^2)$ and in $O(n^3)$, though both are in **P**. Moderately hard functions are not well studied but are indeed useful in some situations.

This chapter is organized as follows: Sections 4.2 and 4.3 are the introduction to CPU-bounded moderately hard functions and memory-bounded counterparts; the fourth and the fifth sections contain applications of these ones. The last section gives a formal definition.

4.2 CPU-bounded functions

Moderately hard functions can be implemented in two ways: CPU-bounded and memory-bounded. The former emphasizes on time while the later focuses on space. The goal of moderately hard functions is to impose extra costs on computation. It can be accomplished by both means but CPU-bounded ones have their limitation.

When one computes a CPU-bounded function, the majority of work is done by the central processor alone. The speed of CPU decides how fast we can compute. Moderately hard functions aim to slow down a computation, not indefinitely but long enough to feel the delay. As CPU chip technology advances, the slowness of moderately hard CPU-bounded functions gets insignificant.

The CPU-bound may still be an effective means if a new generation CPU makes little progress in processing non-parallelized tasks. The CPU clock rate is an index of processor performance. In year 2000, Vikas Agarwal et al. predicted that the annual clock rate improvement would be $12 \backsim 17\%$, compared to $50 \backsim 60\%$ growth in the past [AHKB00]. Indeed, the CPU design trend has focused on multi-core architectures [BDKKMPR05, SC10].

To design moderately hard compute-bound functions against multi-core or many-core CPUs, it is important to make them non-amortized. The final result should be derived from a series of computations. Each intermediate result should be based on

previous ones and none of them can be cached or be omitted.

4.3 Memory-bounded functions

Memory-bounded functions impose extra costs for computation in terms of space. Execution of software programs needs space to store intermediate data and/or instructions. A program is composed of data and instructions. They are originally stored on disk. When the program is loaded, they are copied to main memory. As the processor executes the program, related instructions and data will be read/write from/to main memory.

Main memory is separated from CPU chip in hardware design. It takes time to transfer data and instructions between them. The CPU accesses data from registers, located in the CPU chip, almost 100 times faster than from main memory [BO03]. The latency becomes a bottleneck as CPU technology advances and memory unit cost drops. To deal with this issue, system designers made another kind of memory, cache, to store temporary data to be possibly used in the near future. While the main memory is made of dynamic random access memory(DRAM), cache is implemented with static random access memory(SRAM) and runs faster.

Cache acts as a buffer to shorten the memory data access latency. There are two types of caches, L1 and L2. L1 is smaller in size and runs as fast as registers. L2 is 5 to 10 times slower but still runs 5 to 10 times faster than accessing main memory [BO03]. Figure 4.1 shows the hierarchy of memory. The higher the rank, the faster the access time and hence costlier.

Unlike CPU performance advanced fast in the past three decades, the improvements in memory technology is not prominent. The ratio of CPU clock cycle and memory data access latency was 0.3 in 1980 and 220 in 2005, and the gap is still growing [SC10]. To solve this problem, the cache size in current computer archi-

tecture designs has ever increased. Cache misses are expensive, causing delays of hundreds of CPU clock cycles [Manferdelli07]. Such bottleneck in performance is called the Memory Wall.

The Memory Wall affects the development of CPU design. From 1990 to 2000, microprocessors have been improving in overall performance at a rate of approximately 50∽60% per year [AHKB00]. Performance can be evaluated in terms of CPU clock speed, measured in megahertz (MHz) or gigahertz (GHz). Such rate has decreased greatly in recent years. Instead, the main stream of processor design focuses on multi-core CPUs. The change is partly due to the Memory Wall problem [BDKKMPR05].

Memory-bounded functions are designed according to the Memory Wall problem. The major speedup of computer performance, in terms of running a job which cannot be parallelized, is due to a faster CPU rather than memory. When a function designed to be data-access-driven, it will not benefit as much using faster CPU.

The growing divergence between memory data access latency and CPU speed has important applications for moderately hard functions.

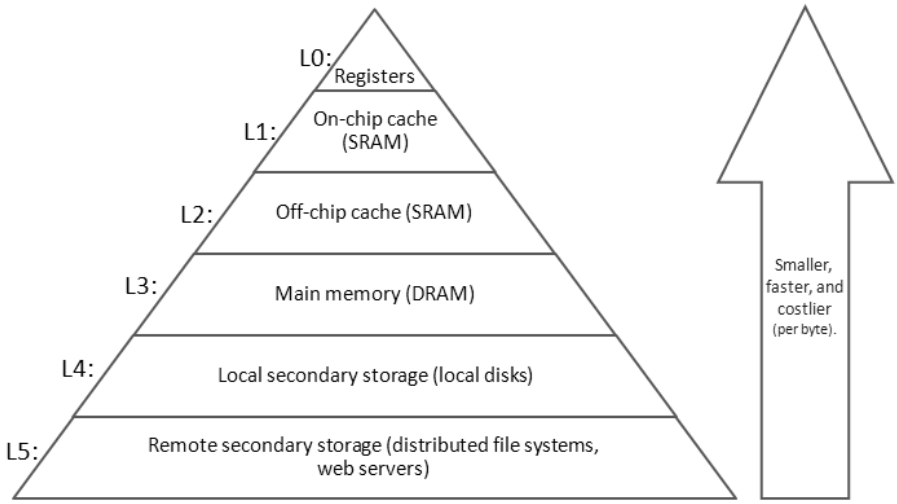

Figure 4.1: A memory hierarchy [BO03]

4.4 Key derivation functions

A password is an important component of a secure computer system. It is an array of 8 or more characters including letters and decimal digits. It should be easily memorized. To gain access to a computer account, a password is usually required. We can also use a password to encrypt electronic files. In that case, a password does not act directly as an encryption key for a symmetric encryption algorithm such as AES. Instead the encryption key, which is much longer than a password, is generated from the password by a *key derivation function*. This procedure is particularly important to protect files on laptops, as lost laptops are posed as serious security risks nowadays. Passwords are usually the weakest link in a cryptographic system, since average users of computer systems tend to select passwords which are vulnerable to dictionary attacks.

If laptops are lost to malicious users, they can obtain encrypted file easily. Attackers can then launch off-line exhaustive search attacks on passwords. To slow down the attacks, we need to make sure that the key derivation function takes a while to output a secret key from a password. Key derivation functions are also used to store passwords in a central server. In a system break-in, attackers can obtain the values outputted by a key derivation function from passwords. It is hoped that since key derivation functions are slow, exhaustive searches for passwords would be very expensive.

A common practice of designing key derivation functions is to apply the popular hash functions like MD5 or SHA1 on passwords recursively for several thousand times. As processors speed up, passwords become weaker at the same rate. Once again we see that using space efficient hash functions like MD5 or SHA1 defeats the purpose of achieving security against attackers with improving computer systems.

Algorithm 3 resolves this issue. If we use M as a key derivation function, then we simply let $M(P)$ be the secret key, where P is the password. To save the password on

a server for identity verifications, we store $M(P)$. We choose l such that the function will run for three seconds on middle range computers. By doing that, we can be sure that low-end systems will not suffer a lot in computing $M(P)$, while high-end systems will not gain much in computing $M(P)$. Exhaustive searching for P given $M(P)$ will be infeasible even for high-end machines.

4.5 Fighting spams

A large percentage of emails we receive are spams. This, ironically, is due to the low cost and efficiency associated with the email system, the properties that first made it so popular. To alleviate this problem, Dwork and Naor in 1992 proposed to charge senders postage of computation efforts [DN92]. The idea has been implemented in the HASHCASH proof-of-work system. Suppose that a sender wants to send us a message m that includes her address, our address, the date and the content, etc. We will require her to attach a string k in the email so that the hash value of m and k putting together, i.e. $h(m, k)$, starts with certain a number of zeros. The sender will have to exhaustively search for the string k, if the hash function can be modeled as a random oracle. The verification part is simple and fast.

If we use the popular hash functions like MD5 or SHA1, which are designed to be space and time efficient, attackers who possess high-end systems with faster processors will have not much difficulty in finding k, if we do not want legitimate senders with only low-end systems to suffer tremendously. On the other hand, if the functions used in finding proof-of-work are space inefficient, not only do we slow the high-end machines that are spreading spam, but we also treat low-end machines more fairly, which often need to send many legitimate emails. In light of this, researchers [ABMW03, DGN03] started to investigate space inefficient hash functions that will access the main memory frequently and randomly during the computation.

Abadi et al. [ABMW03] proposed functions based on inverting of prescribed functions. To prevent the time-space tradeoff attack, Dwork et al. [DGN03] proposed to use functions that read random positions in a big table (for the current configuration, a table of size 16 Mbytes is good enough). The table has to look random. One way to achieve that is to fix a truly random string, i.e. a bit string with high Kolmogorov complexity. But a long random string will greatly increase the size of email client programs. After all, adding 16 Mbytes to email clients is not very appealing. Can we generate such a table using a short program that runs in large space? Dwork et al. [DNW05] designed a graph theoretic method to generate large tables from short inputs, but the procedure is quite involved.

The Dwork et al.'s space inefficient hash function M is built on hash functions H_0, H_1, H_2 and H_3, and a large array T. T is a big table and cannot be entirely placed into cache. Algorithm 3 is copied from [DNW05] for completeness.

Algorithm 4.1: The space inefficient hash function M [DNW05]

Input: m, l
$A = H_0(m)$;
while $l \neq 0$ **do**
  $c = H_1(A)$;
  $A = H_2(A, T[c])$;
  $l = l - 1$;
end
return $H_3(A)$

The time of accessing memory should be the dominating part of the computation, so the hash functions H_1 and H_2 in the above program need to be time and space efficient. The size of A need to be determined carefully. See [DGN03] for details. As for H_0 and H_3, it is a good idea to use MD5 or SHA1. Even though they have weakness in collision resistance, it does not affect this application.

For fighting spam, an email sender is required to attach a bit string k to her message m such that a certain number of bits at the beginning of $M(m, k)$ are all

zeros. We select a number l so that the time to find a k takes about 10 seconds in a middle range computer. We should not require every sender to attach a proof-of-work. Doing this would impose too much overhead on email systems. We note that most of legitimate emails we receive are from addresses that have sent us legitimate emails before. Therefore we need to maintain a list of trusted addresses and domains, and require only the sender from an address not in the list to attach a proof-of-work. The procedure needs to be handled automatically by the system. This greatly reduces the workload of some centralized email servers like Gmail. If combined with other tools such as filters, we believe that it will significantly reduce the amount of spams.

4.6 Formal definition of memory-bounded functions

Since we want the output of a memory-bounded function to look random, it is natural to first examine pseudo-random generators, which are one of the most important cryptographic primitives. There is vast literature on the topic. But most of pseudo-random number generators are not good for the purpose, because given short random inputs, they can be computed in a space efficient manner. Furthermore, most of pseudo-random generators are based on hash chains and they suffer from the time-space tradeoff attacks. Let h be a cryptographic hash function like MD5 or SHA1. Let r_0 be a random string. We can build a hash chain of length e by defining

$$r_i = h(r_{i-1}), 0 \leq i \leq e-1.$$

How fast can we compute r_i for a random i? If we have about one unit of space (assume that each unit can hold one value in the chain), then we have to apply the hash function i times to compute r_i. On average, it takes $O(e)$ hash applications to compute a random element in the hash chain. On the other hand, given s memory

units, we can pre-compute a table

$$r_0, r_{\lfloor e/s \rfloor}, r_{\lfloor 2e/s \rfloor}, \cdots, r_{\lfloor (s-1)e/s \rfloor},$$

which may be regarded as hints for computing the hash chain. Then on average it takes only $O(e/s)$ hash applications to compute an element in the chain. More space allows a more efficient algorithm to compute an element in the chain. This is an example of time-space trade-off.

Time-space trade-off attack is not acceptable to memory-bounded functions. Instead, if space is below a certain point (close to the size of the output), no polynomial time algorithm should exist to compute any part of the output. This motivates the following definition of memory-bounded function.

Definition 4.6.1. We call a function $\mathbf{F} : \{0,1\}^r \to (\{0,1\}^s)^e$ memory-bounded, if

- the function can be computed in time polynomial in r;
- there exists a subset S of $\{0,1\}^r$ with cardinality greater than $2^r - 2^{r/2}$, such that for any function $h : \{0,1\}^r \to (\{0,1\}^s)^{e/2}$, for any polynomial time algorithm A running in space of $se/2 + \log e$ bits and for all $a \in S$, we have

$$\left| \left\{ i \mid 1 \le i \le e, A(h(a), i) = \mathbf{F}(a)[i] \right\} \right| \le \frac{e}{2} + \frac{e}{s}.$$

Remark: Inevitably there are inputs a of $\mathbf{F}$ that $\mathbf{F}(a)[i]$ can be computed in space efficient manner for any i, but the number of them should be small. In the above definition, the number is less than $2^{r/2}$.

Remark: Elements in output array usually belong to the set $\{0, 1, 2, \cdots, q-2, q-1\}$ where $2^{s-1} \le q \le 2^s$. So a $se/2$-bit data can potentially hold $e/2 + e/s$ elements.

Remark: There is no requirement for the computational complexity of the function h. One should consider $h(a)$ as hints, which may be pre-computed to facilitate a space efficient algorithm to compute $\mathbf{F}(a)[i]$.

Remark: If $h(a)$ consists of part of the output array $\mathbf{F}(a)$, then we can read $e/2 + e/h$ elements in the array. But by definition, there is no algorithm that can compute one more element in the limited space of $se/2$ bits.

To summarize the definition, even though the output is generated from a short input, it behaves like a random string to a machine with only limited space.

Theorem 4.6.2. *Assume that H_0, H_1, H_2, H_3 are random oracles in the algorithm defining M. If we implement T by using the output generated by a memory-bounded function $\mathbf{F}$ with a random input of s bits and with $l > r$, the algorithm M will have $l/3$ many cache misses with probability greater than $1 - 0.95^l$.*

Intuitively during the computation of M, it is unlikely that the value of A will repeat. The location of access, c, cannot be predicted if the hash function H_1 can be modeled as a random oracle. If the cache has size $se/2$ or less, then the probability of cache miss would be at least $1/2 - 1/s$, since at most $e/2 + e/s$ many elements in T can be computed using only caches.

Proof. If an input a of $\mathbf{F}$ was selected randomly, then with probability $1 - \frac{1}{2^{r/2}}$, $a \in S$. In any of the l iterations of the loop of M, assume that cache content is C at the beginning of the iteration and the computer is running an algorithm A to compute $\mathbf{F}(a)[c]$. If there is no cache miss, then

$$c \in \Big\{ i \mid 1 \leq i \leq e, A(C, i) = \mathbf{F}(a)[i] \Big\}.$$

Since c is computed from a random oracle H_1, according to definition of memory-bounded functions, no cache miss happens with probability less than or equal to

$1/2+1/s$. For l iterations, the algorithm has $l/3$ or less cache misses with a probability of

$$\sum_{1\le i\le l/3} \binom{l}{i} (1/2+1/s)^{l-i}(1/2-1/s)^i < \frac{\binom{l}{\lceil l/3\rceil}}{2^l} < 0.944^l.$$

The algorithm has $l/3$ or more cache miss with probability at least $(1-0.707^r)(1-0.944^l) > 1-0.95^l$. □

Chapter 5

Number Theoretic Constructions

5.1 Motivation

Number theoretic functions form the backbone of public-key cryptosystems. Computationally cryptography related number theoretic functions are mostly modular exponentiations, or their variations, such as elliptic curve point multiplications. A large amount of work has been done to improve the efficiency of modular exponentiations, but they are notoriously slow, comparing with symmetric encryption/decryption and popular hash functions. On the other hand, usually modular exponentiations can be evaluated in a small amount of space, hence they are not memory-bounded. For instance $a^b \pmod n$ (a, b, n are positive integers and $a, b < n$) can be computed in $O(\log n)$ space, and the output size is not larger than the input size.

5.2 Construction

We consider the exponentiation $(1+x)^n \pmod{p, x^e - a}$. If

$$(1+x)^n \pmod{p, x^e - a} = c_0 + c_1 x + c_2 x^2 + \cdots + c_{e-1} x^{e-1}, \tag{5.2.0.1}$$

we formally define $F_{p,e,a}$ to be

$$F_{p,e,a}(n) = (c_0, c_1, \cdots, c_{e-1}) \in \mathbf{F}_p^e.$$

First observe that an input has size $O(\log p)$ bits if we require that $n < p$ and that e is about $(\log p)^{O(1)}$. The function can be computed in polynomial time using $O(e \log p)$-bit space. However there is no known efficient algorithm to compute any part of coefficient using less than $e \log p$ space. The condition that $x^e - a$ is irreducible is crucial here, for otherwise the ring $\mathbf{F}_p[x]/(x^e - a)$ can be split, and the computation can be done in smaller space.

Notice that the i-th coefficient can indeed be computed in smaller space by evaluating the expression

$$c_i = \sum_{ej+i \le n} \binom{n}{ej+i} a^j \pmod{p}$$

term by term. But the method will take exponential time, because not only the number of terms is exponential in $\log n$, but also each term involves factorial-like functions, which are hard to compute individually.

Conjecture 5.2.1. *The function $F_{p,a,e}$ is memory-bounded.*

One might ask whether for a small τ, there exist hints $h_1, h_2, \cdots, h_\tau \in \mathbf{F}_p$ depending on n, and an algorithm that, given i, computes the c_i in (5.2.0.1) from i and $h_1, h_2, \cdots, h_\tau$. The following lemma shows that this is impossible when there is no restriction on n.

Theorem 5.2.2. *Denote the order of $1+x$ in the field $\mathbf{F}_p[x]/(x^e - a)$ by $ord(1+x)$. Let τ be an integer less than $\log_p ord(1+x)$. For any functions $f_0, f_1, \cdots, f_{e-1}$, there do not exist functions $h_1, h_2, \cdots, h_\tau$ in $\mathbf{Z}^+_{\le p^e} \to \mathbf{F}_p$ such that*

$$(1+x)^n \pmod{p, x^e - a}$$

$$= \sum_{0 \le i \le e-1} f_i(h_1(n), h_2(n), \cdots, h_\tau(n)) x^i$$

for all $1 \le n \le p^e$.

Proof. The theorem can be proved by a counting argument. Fix any function $f_0, f_1, \cdots, f_{e-1}$ and $h_1, h_2, \cdots, h_\tau$,

$$\begin{aligned} & \left| \Big\{ \sum_{0 \le i \le e-1} f_i\Big(h_1(n), h_2(n), \cdots, h_\tau(n)\Big) x^i \Big\} \right| \\ \le & \left| \Big\{ (\hat{h}_1, \hat{h}_2, \cdots, \hat{h}_\tau) \mid \hat{h}_i \in \mathbf{F}_p \text{ for } 1 \le i \le \tau \Big\} \right| \\ = & \ p^\tau < ord(1+x), \end{aligned}$$

but

$$\left| \Big\{ (1+x)^n \pmod{p, x^e - a} \mid 1 \le n \le p^e \Big\} \right| = ord(1+x),$$

thus

$$(1+x)^n \pmod{p, x^e - a} =$$

$$\sum_{0 \le i \le e-1} f_i(h_1(n), h_2(n), \cdots, h_\tau(n)) x^i$$

can not hold for all $1 \le n \le p^e$. □

Remark: Usually the order of $1 + x$ is close to p^e, hence the size of hints has to be very large in order for an algorithm to compute the coefficient in a space efficient manner. The theorem does not prove that F is memory-bounded, because we require that $n \le p$ in F. But it does serve as a support evidence to the conjecture that F is memory-bounded.

5.3 Memory-efficient exponentiating method

In Section 5.2, modular arithmetics refers to calculating in/between equivalent classes according to the modulus. Let $a, q, m, b \in \mathbb{Z}$, $a = qm + b$, where q is quotient and $0 \le b < m$ is residue. The modulus m partitions $\mathbb{Z}$ into m equivalent classes. $a \equiv b \pmod{m}$ means that a and b are equivalent.

Modular exponentiation corresponds to repeated multiplication of the same base for exponent number of times. The product is an integer between 0 and m so does any intermediate result. Modular exponentiation has lots of applications in the field of cryptography. Take the famous RSA scheme as example, the ciphertext $c \equiv m^e \pmod{n}$, where m is the plaintext and n, e together is the public key. Both encryption and decryption rely on modular exponentiation. The speed of exponentiation determines practicability of these schemes.

The naive way of deriving $b^e \pmod{m}$ is first calculate b^e and then modulo it by m. Say we are interested in finding $7^{50} \pmod{19}$. The first step results in

$$179846504264741214662028034056964934925 1249,$$

which needs 43 decimal digits of space to proceed next step while 7 and 50 need at most 2. The space requirement of this method is $O(e \log b)$ and time is $O(e)$.

Observing the fact below, exponentiation can be divided into multiple square steps. And we can apply modulo on each intermediate product to reduce the space requirement.

Observation 1.

$$\begin{aligned} & b^2 \pmod{m} \\ \equiv & \big(b \pmod{m}\big)\big(b \pmod{m}\big) \pmod{m} \\ \equiv & \big(b \pmod{m}\big)^2 \pmod{m} \end{aligned}$$

Integers are represented in binary format in computers. Multiplication by 2 can be done efficiently by just shifting n bit to the left, where n is the number of bits required to represent that integer. Squaring requires only n bit-operations rather

than multiplying. Shifting n-bits can be done faster than multiply an integer by 2. According to this, the following algorithm dramatically reduces time and space requirement. Compared to the intuitive method, it does $O(\log e)$ multiplications and needs $O(\log m)$ in space.

Algorithm 5.1: Exponentiating by squaring [Schneier96]

Input: base b, exponent e, modulus m
Output: *result*

```
1 result := 1;
2 while e > 0 do
3 |   if (e & 1) == 1 then
4 |   |   result := (result * b) (mod m);
5 |   end
6 |   e := e >> 1;
7 |   b := (b * b) (mod m);
8 end
9 return result
```

This technique applies on the situation $(1+x)^n \pmod{p, x^e - a}$, where p is a prime number and n, e, a are in $\mathbb{Z}$. The time requirement is $O(\log n)$ and space is $O(e \log p)$. For example, $(1+x)^{50} \pmod{79, x^{26} - 3}$ is congruent to

$$\begin{aligned} &57x^{25} + 73x^{24} + 63x^{23} + 57x^{22} + 77x^{21} + 11x^{20} + 20x^{19} + 32x^{18} + 28x^{17} \\ &+61x^{16} + x^{15} + 46x^{14} + 68x^{13} + 17x^{12} + 72x^{11} + 78x^{10} + 77x^9 + 7x^8 \\ &+9x^7 + 32x^6 + 62x^5 + 71x^4 + 9x^3 + 9x^2 + 26x + 53. \end{aligned}$$

It is hard to derive the coefficient of certain term without using $O(e \log p)$ space. For each iteration of while loop, the algorithm squares base and then does modulo. These intermediate results take up space.

5.4 Closed form observation

One criterion of choosing a memory bounded function is to show that there is no shortcut way to derive any partial results. In the case of $(1+x)^n \pmod{p, x^e - a}$, we are looking for evidence of deriving some binomial coefficients using small memory space. $(1+x)^n \pmod{p, x^e - a}$ will be a good candidate of memory bound if we cannot find such certificate.

The binomial formula $(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$ relates to our goal. We are interested in computing sums of binomial coefficients in small space, or calculate it directly, where k is not consecutive integers. More generally, we consider the case of Definition 5.4.3.

Definition 5.4.1. $\forall n, k \in \mathbb{Z}$, $\binom{n}{k} \stackrel{\text{def}}{=} \frac{n(n-1)(n-2)\cdots(n-k+1)}{k!}$.

Definition 5.4.2. $\forall n, k \in \mathbb{Z}$, $\binom{n}{k} \stackrel{\text{def}}{=} 0$ if $k < 0$, $k > n$, or $n \leq 0$.

Definition 5.4.3.

$$f_i(n,l) \stackrel{\text{def}}{=} \sum_{0 \leq k \leq n} \binom{n}{lk-i}, \text{ where } i, l \in \mathbb{N} \text{ and } 0 \leq i \leq l.$$

One way to compute $f_i(n,l)$ in small space is to express it in closed form. For a fixed n and i, $f_i(n,l)$ becomes a series as l varies. A closed form exhibits a series as a sum of a fixed (independent of n) number of hypergeometric terms.

A *geometric* series $\sum_{k \geq 0} t_k$ is one in which the ratio of every two consecutive terms is constant, i.e., $\frac{t_{k+1}}{t_k}$ is a constant function of the summation index k. A *hypergeometric* series $\sum_{k \geq 0} t_k$ is one in which $t_0 = 1$ and the ratio of two consecutive terms is a *rational function* of the summation index k, i.e., in which

$$\frac{t_{k+1}}{t_k} = \frac{P(k)}{Q(k)},$$

where P and Q are polynomials in k. Examples of such hypergeometric terms are $t_k = x^{2k}$ or $t_k = \frac{(3k-8)!}{(k+1)!}$.

Definition 5.4.4 (Hypergeometric closed form [PWZ96])**.** A function $f(n)$ is said to be of *closed form* if it is equal to a linear combination of a fixed number, r, say, of hypergeometric terms. The number r must be an absolute constant, i.e., it must be independent of all variables and parameters of the problem.

Petkovsek et al. [PWZ96] developed computer programs for simplifying sums that involve binomial coefficients. Their algorithm, Hyper, outputs all hypergeometric closed forms of a recursive relation. A recurrence relation is an equation that recursively defines a sequence: each term of the sequence is defined as a function of the preceding terms.

We are going to find a recursive relation for $f_i(n, l)$ and start with a special case.

Definition 5.4.5.

$$f_i(n) \stackrel{\text{def}}{=} \sum_{0 \le k \le n} \binom{n}{3k-i}.$$

Observation 2.

$$f_0(n+1) = f_0(n) + \sum_{0 \le k \le n} \binom{n}{3k-1} = f_0(n) + f_1(n).$$
$$f_1(n+1) = f_1(n) + \sum_{0 \le k \le n} \binom{n}{3k-2} = f_1(n) + f_2(n).$$
$$f_2(n+1) = f_2(n) + \sum_{0 \le k \le n} \binom{n}{3k} = f_2(n) + f_0(n).$$

Lemma 5.4.6. $2f_i(n) - 3f_i(n+1) + 3f_i(n+2) - f_i(n+3) = 0$ *is the recursive relation of* $f_i(n)$.

Proof. The proof is done by induction on n.

1. When $n = 1$, we have

$$2f_0(1) - 3f_0(2) + 3f_0(3) - f_0(4) = 2 - 3 + 6 - 5 = 0.$$
$$2f_1(1) - 3f_1(2) + 3f_1(3) - f_1(4) = 0 - 3 + 9 - 6 = 0.$$
$$2f_2(1) - 3f_2(2) + 3f_2(3) - f_2(4) = 2 - 6 + 9 - 5 = 0.$$

2. Assume

$$2f_0(n) - 3f_0(n+1) + 3f_0(n+2) - f_0(n+3) = 0 \text{ and}$$
$$2f_1(n) - 3f_1(n+1) + 3f_1(n+2) - f_1(n+3) = 0 \text{ and}$$
$$2f_2(n) - 3f_2(n+1) + 3f_2(n+2) - f_2(n+3) = 0.$$

3. Want to show $2f_0(n+1) - 3f_0(n+2) + 3f_0(n+3) - f_0(n+4) = 0$.

$$\begin{aligned}
&2f_0(n+1) - 3f_0(n+2) + 3f_0(n+3) - f_0(n+4) \\
=&2\left(f_0(n) + \sum_{0\le k\le n}\binom{n}{3k-1}\right) - 3\left(f_0(n+1) + \sum_{0\le k\le n+1}\binom{n+1}{3k-1}\right) \\
&+ 3\left(f_0(n+2) + \sum_{0\le k\le n+2}\binom{n+2}{3k-1}\right) - \left(f_0(n+3) + \sum_{0\le k\le n+3}\binom{n+3}{3k-1}\right) \\
=&2\sum_{0\le k\le n}\binom{n}{3k-1} - 3\sum_{0\le k\le n+1}\binom{n+1}{3k-1} + 3\sum_{0\le k\le n+2}\binom{n+2}{3k-1} - \sum_{0\le k\le n+3}\binom{n+3}{3k-1} \\
=&2f_1(n) - 3f_1(n+1) + 3f_1(n+2) - f_1(n+3) \\
=&2f_1(n+1) - 3f_1(n+2) + 3f_1(n+3) - f_1(n+4) \\
=&2\left(f_1(n) + \sum_{0\le k\le n}\binom{n}{3k-2}\right) - 3\left(f_1(n+1) + \sum_{0\le k\le n+1}\binom{n+1}{3k-2}\right) \\
&+ 3\left(f_1(n+2) + \sum_{0\le k\le n+2}\binom{n+2}{3k-2}\right) - \left(f_1(n+3) + \sum_{0\le k\le n+3}\binom{n+3}{3k-2}\right)
\end{aligned}$$

$$
\begin{aligned}
=&2\sum_{0\le k\le n}\binom{n}{3k-2}-3\sum_{0\le k\le n+1}\binom{n+1}{3k-2}+3\sum_{0\le k\le n+2}\binom{n+2}{3k-2}-\sum_{0\le k\le n+3}\binom{n+3}{3k-2}\\
=&2f_2(n)-3f_2(n+1)+3f_2(n+2)-f_2(n+3)\\
=&2f_2(n+1)-3f_2(n+2)+3f_2(n+3)-f_2(n+4)\\
=&2\left(f_2(n)+\sum_{0\le k\le n}\binom{n}{3k}\right)-3\left(f_2(n+1)+\sum_{0\le k\le n+1}\binom{n+1}{3k}\right)\\
&+3\left(f_2(n+2)+\sum_{0\le k\le n+2}\binom{n+2}{3k}\right)-\left(f_2(n+3)+\sum_{0\le k\le n+3}\binom{n+3}{3k}\right)\\
=&2\sum_{0\le k\le n}\binom{n}{3k}-3\sum_{0\le k\le n+1}\binom{n+1}{3k}+3\sum_{0\le k\le n+2}\binom{n+2}{3k}-\sum_{0\le k\le n+3}\binom{n+3}{3k}\\
=&2f_0(n)-3f_0(n+1)+3f_0(n+2)-f_0(n+3)\\
=&0
\end{aligned}
$$

□

Lemma 5.4.6 can be further generalized.

Lemma 5.4.7.

$$
\begin{aligned}
f_0(n+1,l)&=f_0(n,l)+f_1(n,l).\\
f_1(n+1,l)&=f_1(n,l)+f_2(n,l).\\
f_2(n+1,l)&=f_2(n,l)+f_3(n,l).\\
&\vdots\\
f_{l-1}(n+1,l)&=f_{l-1}(n,l)+f_0(n,l).
\end{aligned}
$$

Proof. According to Pascal's rule, the proof is straight forward.

$$
f_0(n+1,l)=\sum_{0\le k\le n}\binom{n}{lk}+\sum_{0\le k\le n}\binom{n}{lk-1}=f_0(n,l)+f_1(n,l).
$$

$$
\begin{aligned}
f_1(n+1,l) &= \sum_{0\le k\le n}\binom{n}{lk-1} + \sum_{0\le k\le n}\binom{n}{lk-2} = f_1(n,l)+f_2(n,l).\\
f_2(n+1,l) &= \sum_{0\le k\le n}\binom{n}{lk-2} + \sum_{0\le k\le n}\binom{n}{lk-3} = f_2(n,l)+f_3(n,l).\\
&\vdots\\
f_{l-1}(n+1,l) &= \sum_{0\le k\le n}\binom{n}{lk-l+1} + \sum_{0\le k\le n}\binom{n}{lk} = f_{l-1}(n,l)+f_0(n,l).
\end{aligned}
$$

□

Definition 5.4.8 (Operator N)**.** $\forall k\in\mathbb{Z}$, $f_i(n,l)N^k \stackrel{\text{def}}{=} f_i(n+k,l)$ and $f_i(n,l)/N^k \stackrel{\text{def}}{=} f_i(n-k,l)$.

Lemma 5.4.9. *$f_i(n,l)\left(1+(1-N)^l\right)=0$ is the recursive relation of $f_i(n,l)$, where $2\nmid l$.*

Proof. The proof is done by induction on n.

1. When $n=0$, we have

$$
f_i(0,l)\left(1+(1-N)^l\right) = \left(\sum_{0\le k\le n}\binom{0}{lk-i}\right)\left(1+(1-N)^l\right) = 0,\forall i.
$$

2. Assume $f_i(n,l)\left(1+(1-N)^l\right)=0,\forall i$.

3. $f_0(n+1,l)\left(1+(1-N)^l\right) = \left(f_0(n,l)+f_1(n,l)\right)\left(1+(1-N)^l\right) = 0$. Similarly, it holds for all other i.

□

Lemma 5.4.10. *$\frac{n}{N}f_i(n,l)\left(1-(1-N)^l\right) = 0$ is the recursive relation of $f_i(n,l)$, where $2\mid l$.*

Proof. The proof is done by induction on n.

1. When $n = 0$, we have

$$\frac{0}{N} f_i(0, l) \left(1 - (1 - N)^l\right) = 0, \forall i.$$

2. Assume $\frac{n}{N} f_i(n, l) \left(1 - (1 - N)^l\right) = 0, \forall i$.

3. $\frac{n+1}{N} f_0(n + 1, l) \left(1 - (1 - N)^l\right) = \frac{n+1}{N} \left(f_0(n, l) + f_1(n, l)\right) \left(1 - (1 - N)^l\right)$
   $= 0$. Similarly, it holds for all other i.

□

The process of deriving closed form solutions of a function containing binomial coefficients involves two steps [PWZ96]:

- Step 1: Convert a summand function into recurrence.
- Step 2: Find all closed form solutions for the recurrence.

The first step is done by Zeilberger's algorithm, which fast discovers the recurrence for a proper hypergeometric term. The second part is finished by the algorithm Hyper. They have been implemented as Maple and Mathematica programs respectively. Table 5.1 shows the inputs and outputs of Zeilberger's algorithm, program *ct*. In case of $y(n) = \sum \binom{n}{8k}$ and higher, the program *ct* returns the following: *Error, (in ct) cannot determine if this expression is true or false: FAIL < 0.*

Figure 5.1 shows outputs from the algorithm Hyper by plugging first five recurrences from above table. As l increases, the level of recurrence increases, which means higher space complexity.

When Hyper returns the empty brackets "{}", it signifies the absence of hypergeometric solutions. If it returns 2, for instance, the answer corresponds to $y(n) = 2^n$.

```
In[68]:= Hyper[-y[n + 3] + 3 y[n + 2] - 3 y[n + 1] + 2 y[n] == 0, y[n], Solutions → All]

N::meprec :
 Internal precision limit $MaxExtraPrecision = 50.` reached while evaluating 3 (1 - (-1)^(1/3) + (-1)^(2/3)). More...

N::meprec :
 Internal precision limit $MaxExtraPrecision = 50.` reached while evaluating 3 (1 - (-1)^(1/3) + (-1)^(2/3)). More...

Out[68]= {2}

In[70]:= Hyper[-y[n + 3] n + 4 y[n + 2] n - 6 y[n + 1] n + 4 y[n] == 0, y[n], Solutions → All]

Out[70]= {}

In[71]:= Hyper[-y[n + 5] + 5 y[n + 4] - 10 y[n + 3] + 10 y[n + 2] - 5 y[n + 1] + 2 y[n] == 0, y[n], Solutions → All]

Out[71]= {2, 1/4 (3 - Sqrt[5] - I Sqrt[2 (5 - Sqrt[5])]), 1/4 (3 - Sqrt[5] + I Sqrt[2 (5 - Sqrt[5])]),
  1/4 (3 + Sqrt[5] - I Sqrt[2 (5 + Sqrt[5])]), 1/4 (3 + Sqrt[5] + I Sqrt[2 (5 + Sqrt[5])])}

In[72]:= Hyper[-y[n + 5] n + 6 y[n + 4] n - 15 y[n + 3] n + 20 y[n + 2] n - 15 y[n + 1] n + 6 y[n] == 0, y[n],
 Solutions → All]

N::meprec : Internal precision limit $MaxExtraPrecision =
   50.` reached while evaluating -(I (<<1>>))/(2 (<<1>>) Sqrt[(-5 - 10 I) (<<1>>)]). More...

N::meprec : Internal precision limit $MaxExtraPrecision =
   50.` reached while evaluating (I (<<1>>))/(2 (<<1>>) Sqrt[(-5 - 10 I) (<<1>>)]). More...

N::meprec : Internal precision limit $MaxExtraPrecision =
   50.` reached while evaluating -(I (<<1>>))/(2 (<<1>>) Sqrt[(-5 - 10 I) (<<1>>)]). More...

General::stop : Further output of N::meprec will be suppressed during this calculation. More...

Out[72]= {}

In[73]:= Hyper[-y[n + 7] + 7 y[n + 6] - 21 y[n + 5] + 35 y[n + 4] - 35 y[n + 3] + 21 y[n + 2] - 7 y[n + 1] + 2 y[n] == 0,
 y[n], Solutions → All]

N::meprec : Internal precision limit $MaxExtraPrecision = 50.` reached
   while evaluating -(-2 + <<1>>) (1 - 3 Root[<<1>> &, 1] + <<6>> + Root[<<1>> &, 1]^6). More...

N::meprec : Internal precision limit $MaxExtraPrecision = 50.` reached
   while evaluating -(-2 + <<1>>) (1 - 3 Root[<<1>> &, 2] + <<6>> + Root[<<1>> &, 2]^6). More...

N::meprec : Internal precision limit $MaxExtraPrecision = 50.` reached
   while evaluating -(-2 + <<1>>) (1 - 3 Root[<<1>> &, 3] + <<6>> + Root[<<1>> &, 3]^6). More...

General::stop : Further output of N::meprec will be suppressed during this calculation. More...

Out[73]= {2}
```

Figure 5.1: Outputs from the algorithm Hyper

Table 5.1: Inputs and outputs of Zeilberger's algorithm

Input	Output
$y(n) = \sum \binom{n}{3k}$	$2y(n) - 3y(n+1) + 3y(n+2) - y(n+3) = 0$
$y(n) = \sum \binom{n}{4k}$	$4ny(n) - 6ny(n+1) + 4ny(n+2) - ny(n+3) = 0$
$y(n) = \sum \binom{n}{5k}$	$2y(n) - 5y(n+1) + 10y(n+2)$ $-10y(n+3) + 5y(n+4) - y(n+5) = 0$
$y(n) = \sum \binom{n}{6k}$	$6ny(n) - 15ny(n+1) + 20ny(n+2)$ $-15ny(n+3) + 6ny(n+4) - ny(n+5) = 0$
$y(n) = \sum \binom{n}{7k}$	$2y(n) - 7y(n+1) + 21y(n+2) - 35y(n+3)$ $+35y(n+4) - 21y(n+5) + 7y(n+6) - y(n+7) = 0$

Hence, for the case of $f_i(n, l)$, we conjecture that there is no closed form solution when $2 \mid l$.

5.5 Conclusion

Popular hash functions are designed to be collision resistant and to be space efficient. Space efficiency is an undesired property in fighting spam and in deriving secret keys from short passwords. We hardly need to worry about collisions in these applications, since passwords are very short, and collisions do not help to decrease the effort to find proof-of-work. We propose to use the space inefficiency of exponentiations of sparse polynomials to build memory-bounded functions. They can be used to design space inefficient hash functions.

Chapter 6

Conclusion and Future Work

The time and space trade-off technique helps a lot in finding lower bound of the sum-of-square-roots problem, a famous open problem in computational geometry. In this dissertation, an upper bound is presented along with a space-efficient algorithm to find $r(n, k)$ exactly in $n^{k+o(k)}$ time and in $n^{\lceil k/2 \rceil + o(k)}$ space. As an example, $r(100, 7)$ is calculated in a few hours on one PC and $r(100, 8)$ in about one month. Numerical data seems to suggest that our upper bound is better than the root separation bounds. Further investigation, both experimental and theoretical, is needed.

Moderately hard functions have been proposed to combat junk emails. One way to implement is by utilizing the memory bound. Memory-bounded functions incorporate random walks in memory space. They use time and space trade-off technique with emphasis on large memory usage and hence properly slow down CPU throughput. High-end computers do not enjoy much acceleration than low-ends. These functions are essentially hash functions with a special constraint.

Popular hash functions are designed to be collision resistant and space efficient. Space efficiency is an undesired property in fighting spam and in deriving secret keys from short passwords. We hardly need to worry about collisions in these applications, since passwords are very short, and collisions do not help to decrease the effort to find proof-of-work. Moderately hard memory-bounded functions can be considered as memory-inefficient hash functions.

A formal definition of memory-bounded functions is given in Chapter 4. This is the first to the best of our knowledge. Hard number theoretic functions form foundations

of modern cryptography. Based on the definition, we aim to construct tables in a number theoretical way.

Chapter 5 demonstrates how to build such tables. Compelling a computer to randomly walk in memory space relies on a table with uniformly distributed data. Such tables should be constructed effortlessly so that they can be shared even though they take lot of memory space. This is an important requirement for practical concerns.

The future goal is to prove a function memory-bounded. We have reasons to conjecture that the constructed table will make a function memory-bounded. It would be nice to find out a proof.

Bibliography

[ABMW03] Martin Abadi, Mike Burrows, Mark Manasse, and Ted Wobber (2003) *Moderately hard, memory-bound functions.* Proceeding of the 10th Network and Distributed System Security Symposium (NDSS), ISBN: 1-891562-16-9.

[AE04] Dana Angluin and Sarah Eisenstat (2004) *How close can $\sqrt{a} + \sqrt{b}$ be to an integer?* Technical Report 1279, Department of Computer Science, Yale Univ, ftp://ftp.cs.yale.edu/pub/TR/tr1279.pdf.

[AHKB00] Vikas Agarwal, M.S. Hrishikesh, Stephen W. Keckler, and Doug Burger (2000) *Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures.* Proceeding of the 27th International Symposium on Computer Architecture, ISSN: 1063-6897, pages 248-259.

[BDKKMPR05] Shekhar Y. Borkar, Pradeep Dubey, Kevin C. Kahn, David J. Kuck, Hans Mulder, Stephen S. Pawlowski, and Justin R. Rattner (2005) *Platform 2015: Intel Processor and Platform Evolution for the Next Decade.* http://www.intel.com/technology/architecture/platform2015/.

[Bernstein01] Daniel Bernstein (2001) *Enumerating solutions to $p(a) + q(b) = r(c) + s(d)$.* Mathematics of Computation, ISSN: 0025-5718, volume 70, number 233, pages 389-394.

[BFMS00] Christoph Burnikel, Rudolf Fleischer, Kurt Mehlhorn, and Stefan Schirra (2000) *A Strong and Easily Computable Separation Bound for Arithmetic Expressions Involving Radicals.* Algorithmica, ISSN: 0178-4617, volume 27, number 1, pages 87-99.

[BLP03] Peter Borwein, Petr Lisoněk, and Colin Percival (2003) *Computational Investigations of the Prouhet–Tarry–Escott Problem.* Mathematics of Computation, ISSN: 0025-5718, volume 72, number 244, pages 2063-2070.

[BO03] Randal E. Bryant and David R. O'Hallaron (2003) *Computer Systems: A Programmer's Perspective.* Prentice Hall. ISBN: 0-13-034074-X.

[Boreico08] Iurie Boreico (2008) *My Favorite Problem: Linear Independence of Radicals.* The Harvard College Mathematics Review, volume 2, number 1.

[Cheng06] Qi Cheng (2006) *On Comparing Sums of Square Roots of Small Integers.* The 31st International Symposium on Mathematical Foundations of Computer

Science (MFCS), Lecture Notes in Computer Science, volume 4162, ISBN: 978-3-540-37791-7, pages 250-255.

[CL08] Qi Cheng and Yu-Hsin Li (2008) *A Number Theoretic Memory Bounded Function and Its Applications.* The International Symposium on Trusted Computing (TrustCom '08), pages 2021-2025, Digital Object Identifier 10.1109/ICYCS.2008.114.

[CL10] Qi Cheng and Yu-Hsin Li (2010) *Finding the Smallest Gap between Sums of Square Roots.* The 9th Latin American Theoretical Informatics Symposium (LATIN '10), Lecture Notes in Computer Science, volume 6034, ISBN: 978-3-642-12199-9, pages 446-455.

[DGN03] Cynthia Dwork, Andrew Goldberg, and Moni Naor (2003) *On memory-bound functions for fighting spam.* Advances in Cryptology-CRYPTO '03, volume 2729, ISBN: 978-3-540-40674-7, pages 426444.

[DMO01] Erik D. Demaine, Joseph S. B. Mitchell, and Joseph O'Rourke (2001) *The Open Problems Project, Problem33: Sum of Square Roots.* http://maven.smith.edu/∼orourke/TOPP/P33.html.

[DN92] Cynthia Dwork and Moni Naor (1992) *Pricing via Processing or Combatting Junk Mail.* Advances in Cryptology-CRYPTO '92, volume 740, ISBN: 978-3-540-57340-1, pages 139-147.

[DNW05] Cynthia Dwork, Moni Naor, and Hoeteck Wee (2005) *Pebbling and proofs of work.* Advances in Cryptology-CRYPTO '05, volume 3621, ISBN: 978-3-540-28114-6, pages 37-54.

[Hellman80] Martin Hellman (1980) *A Cryptanalytic Time-Memory Trade-Off.* IEEE Transactions on Information Theory, ISSN: 0018-9448, volume 26, issue 4, pages 401-406.

[JKR02] Jaeyeon Jung, Balachander Krishnamurthy, and Michael Rabinovich (2002) *Flash crowds and denial of service attacks: characterization and implications for CDNs and web sites.* The 11th international conference on World Wide Web (WWW '02), ISBN:1-58113-449-5, pages 293-304.

[Knuth73] Donald Knuth (1973) *The Art of Computer Programming, volume 3.* Addison-Wesley. ISBN:0-20103-803-X.

[Manferdelli07] John L. Manferdelli (2007) *The Many-Core Inflection Point for Mass Market Computer Systems.* CTWatch Quarterly, ISSN: 1555-9874, February.

[O'Rourke81] Joseph O'Rourke (1981) *Advanced problem 6369.* American Mathematical Monthly, volume 88, number 10, page 769.

[PWZ96] Marko Petkovsek, Herbert Wilf, and Doron Zeilberger (1996) $A = B$. A K Peters Ltd., ISBN: 978-1-56881-063-8.

[QD08] QD v2.3.7 (C++/Fortran-90 double-double and quad-double computation package), http://crd.lbl.gov/∼dhbailey/mpdist/.

[QW06] Jianbo Qian and Cao An Wang (2006) *How much precision is needed to compare two sums of square roots of integers?* Information Processing Letters, ISSN: 0020-0190, volume 100, issue 5, pages 194-198.

[SC10] X.-H. Sun and Y. Chen (2010) *Reevaluating Amdahl's law in the multicore era.* Journal of Parallel and Distributed Computing, ISSN: 0743-7315, volume 70, number 2, pages 183-188.

[Schneier96] Bruce Schneier (1996) *Applied Cryptography: Protocols, Algorithms, and Source Code in C, Second Edition (2nd ed.).* Wiley. ISBN: 978-0471117094.

[Schonhage79] Arnold Schönhage (1979) *On the power of random access machines.* Proceedings of the 6th Colloquium, on Automata, Languages and Programming, Lecture Notes in Computer Science, volume 71, ISBN: 978-3-540-09510-1, pages 520-529.

[Shamir79] Adi Shamir (1979) *Factoring numbers in $O(\log n)$ arithmetic steps.* Information Processing Letters, ISSN: 0020-0190, volume 8, number 1, pages 28-31.

Appendices

Appendix A

Proof of Linear Independence

Definition A.0.1. Let $a_1, a_2, \cdots, a_h \in \mathbb{Z}$ be not all zeros. Define a set of linear expressions

$$L(x_1, x_2, \cdots, x_h) \stackrel{\text{def}}{=} \{a_1 x_1 \pm a_2 x_2 \pm \cdots \pm a_h x_h\},$$

which contains all combinations of sign of each term.

There are 2^{h-1} elements in $L(x_1, x_2, \cdots, x_h)$.

Definition A.0.2. Let $T \in \mathbb{Z}\backslash\{0\}$ be a variable.

$$F_{L,x_1,x_2,x_3,\cdots,x_h}(T) \stackrel{\text{def}}{=} \prod_{e\in L}(T-e) = \prod \Big(T - (a_1 x_1 \pm a_2 x_2 \pm a_3 x_3 \pm \cdots \pm a_h x_h)\Big).$$

Note that changing the sign of any of $x_2, x_3, \cdots, x_h$ only re-permutes the set $L(x_1, x_2, \cdots, x_h)$. So, $F_{L,x_1,x_2,x_3,\cdots,x_h}(T) = F_{L,x_1,\pm x_2,\pm x_3,\cdots,\pm x_h}(T) = \prod(T - a_1 x_1 \pm a_2 x_2 \pm a_3 x_3 \pm \cdots \pm a_h x_h)$. This implies that the power of $x_2, x_3, \cdots, x_h$ are even in the expanded form of $F_{L,x_1,x_2,\cdots,x_h}(T)$.

The power of x_1 can be even or odd. We treat $F_{L,x_1,\cdots,x_h}(T)$ as a polynomial of $x_1, x_2, \cdots, x_h$ and group its expanded form by parity of x_1.

$$F_{L,x_1,\cdots,x_h}(T) = x_1 P(x_1^2, x_2, x_3, \cdots, x_h, T) + Q(x_1^2, x_2, x_3, \cdots, x_h, T),$$

where P contains monomials with power of x_1 is odd and Q does that with even

power. Since the power of $x_2, x_3, \cdots, x_h$ are even in the expanded form, we can write

$$F_{L,x_1,\cdots,x_h}(T) = x_1 P'(x_1^2, x_2^2, x_3^2, \cdots, x_h^2, T) + Q'(x_1^2, x_2^2, x_3^2, \cdots, x_h^2, T).$$

Note that P' and Q' generate integers if $x_1^2, x_2^2, x_3^2, \cdots, x_h^2$, and T are all integers.

Proposition A.0.3. *Let $n_1, n_2, \cdots, n_N$ be square-free numbers and $a_1, a_2, \cdots, a_N \in \mathbb{Z}$ be not all zeros.*

$$\sum_{1 \le i \le N} a_i \sqrt{n_i} \notin \mathbb{Z} \backslash \{0\}.$$

Proof. [Boreico08] Prove by induction on N. When $N = 1$, it is clear. Assume on the contradiction that $a_1\sqrt{n_1} + a_2\sqrt{n_2} + \cdots + a_h\sqrt{n_h} = M \in \mathbb{Z}\backslash\{0\}$. By definition, $L(\sqrt{n_1}, \sqrt{n_2}, \cdots, \sqrt{n_h}) = \{a_1\sqrt{n_1} \pm \cdots \pm a_h\sqrt{n_h}\}$. We have $F_{L,\sqrt{n_1},\sqrt{n_2},\cdots,\sqrt{n_h}}(M) = \prod_{e \in L}(M - e) = 0$. We also have $F_{L,\sqrt{n_1},\sqrt{n_2},\cdots,\sqrt{n_h}}(M) = \sqrt{n_1}P'(n_1, n_2, \cdots, n_h, M) + Q'(n_1, n_2, \cdots, n_h, M) = 0$. So,

$$P'(n_1, n_2, \cdots, n_h, M) = Q'(n_1, n_2, \cdots, n_h, M) = 0.$$

Hence, $-\sqrt{n_1}P'(n_1, n_2, \cdots, n_h, M) + Q'(n_1, n_2, \cdots, n_h, M) = F_{L,-\sqrt{n_1},\sqrt{n_2},\cdots,\sqrt{n_h}}(M) = \prod(M + a_1\sqrt{n_1} \pm a_2\sqrt{n_2} \pm a_3\sqrt{n_3} \pm \cdots \pm a_h\sqrt{n_h}) = 0$.

Therefore, $M = -a_1\sqrt{n_1} \pm a_2\sqrt{n_2} \pm \cdots \pm a_h\sqrt{n_h}$ for some combination of sign. But we have assumed that $M = a_1\sqrt{n_1} + a_2\sqrt{n_2} + \cdots + a_h\sqrt{n_h} \in \mathbb{Z}\backslash\{0\}$. Summation of these two M cancels $a_1\sqrt{n_1}$ and results in

$$2M = (a_2 \pm a_2)\sqrt{n_2} + (a_3 \pm a_3)\sqrt{n_3} + \cdots + (a_h \pm a_h)\sqrt{n_h},$$

which contradicts to the induction hypothesis. □

Proposition A.0.4. *The set $\{\sqrt{n} \mid n \in \mathbb{N}$ is square-free$\}$ is linearly independent over rationals.*

Proof. The proposition is equivalent to the following: Let $a_i \in \mathbb{Z}$ and let n_i be different square-free numbers, where $1 \leq i \leq N$.

If $a_1, a_2, \cdots, a_N$ are not all zeros , then $a_1\sqrt{n_1} + a_2\sqrt{n_2} + \cdots + a_N\sqrt{n_N} \neq 0$.

Prove by induction on N. When $N = 1$, it is clear. Assumption that $a_1\sqrt{n_1} + a_2\sqrt{n_2} + \cdots + a_{h-1}\sqrt{n_{h-1}} \neq 0$. When $N = h$, assume on the contradiction that $a_1\sqrt{n_1} + a_2\sqrt{n_2} + \cdots + a_h\sqrt{n_h} = 0$.

$$a_1\sqrt{n_1} + a_2\sqrt{n_2} + \cdots + a_{h-1}\sqrt{n_{h-1}} = -a_h\sqrt{n_h}$$
$$a_1\sqrt{n_1n_h} + a_2\sqrt{n_2n_h} + \cdots + a_{h-1}\sqrt{n_{h-1}n_h} = -a_hn_h \in \mathbb{Z}\backslash\{0\},$$

which contradicts to the Proposition A.0.3. □

Appendix B

Statistics on Sums

Table B.1: Statistics on the summations of square roots of $(n, k) = (5000, 3)$.

3	**4**	**5**	**6**	**7**	**8**
4	15	43	99	206	400
9	**10**	**11**	**12**	**13**	**14**
713	1185	1911	2950	4385	6337
15	**16**	**17**	**18**	**19**	**20**
8907	12295	16655	22097	28981	37357
21	**22**	**23**	**24**	**25**	**26**
47650	60052	74965	92677	113525	138027
27	**28**	**29**	**30**	**31**	**32**
166551	199577	237948	281279	331464	388240
33	**34**	**35**	**36**	**37**	**38**
452345	524933	606445	697749	799706	913467
39	**40**	**41**	**42**	**43**	**44**
1039529	1178845	1333677	1503114	1690466	1895287
45	**46**	**47**	**48**	**49**	**50**

Continued on Next Page...

Table B.1 – Continued

2119584	2365011	2632038	2923134	3239383	3582228
51	**52**	**53**	**54**	**55**	**56**
3953396	4354689	4788446	5255278	5758272	6299144
57	**58**	**59**	**60**	**61**	**62**
6879535	7501883	8169257	8881764	9644207	10458582
63	**64**	**65**	**66**	**67**	**68**
11325468	12250208	13234034	14279067	15390870	16568833
69	**70**	**71**	**72**	**73**	**74**
17820269	19142611	20545988	22028164	23595388	25249510
75	**76**	**77**	**78**	**79**	**80**
26991603	28829005	30758422	32784371	34912226	37136255
81	**82**	**83**	**84**	**85**	**86**
39464476	41895567	44431313	47070060	49813869	52667301
87	**88**	**89**	**90**	**91**	**92**
55623906	58685743	61857470	65129967	68508590	71988885
93	**94**	**95**	**96**	**97**	**98**
75575207	79258476	83041661	86920838	90896965	94962543
99	**100**	**101**	**102**	**103**	**104**
99116748	103360270	107683773	112086586	116567538	121118278
105	**106**	**107**	**108**	**109**	**110**
125731862	130412102	135150468	139935625	144773478	149645947
111	**112**	**113**	**114**	**115**	**116**

Continued on Next Page...

Table B.1 – Continued

154556626	159492487	164452779	169426215	174404293	179384900
117	**118**	**119**	**120**	**121**	**122**
184354909	189310548	194237568	199129554	203983309	208780264
123	**124**	**125**	**126**	**127**	**128**
213515162	218177924	222757232	227237453	231620252	235881678
129	**130**	**131**	**132**	**133**	**134**
240016469	244004262	247848231	251519386	255011530	258313668
135	**136**	**137**	**138**	**139**	**140**
261403948	264275199	266915487	269297163	271415683	273251496
141	**142**	**143**	**144**	**145**	**146**
274788141	276015820	276924539	277528487	277830401	277833091
147	**148**	**149**	**150**	**151**	**152**
277542224	276956357	276090607	274937063	273509996	271814719
153	**154**	**155**	**156**	**157**	**158**
269852829	267631210	265151762	262424757	259459748	256257600
159	**160**	**161**	**162**	**163**	**164**
252824364	249168646	245299561	241221119	236944794	232477497
165	**166**	**167**	**168**	**169**	**170**
227818206	222986772	217988366	212827174	207518811	202062422
171	**172**	**173**	**174**	**175**	**176**
196479816	190771427	184951056	179026817	173008101	166909684
177	**178**	**179**	**180**	**181**	**182**

Continued on Next Page...

Table B.1 – Continued

160735535	154502639	148219297	141895510	135546660	129181081
183	**184**	**185**	**186**	**187**	**188**
122813342	116453357	110116641	103812122	97558366	91363708
189	**190**	**191**	**192**	**193**	**194**
85241778	79210217	73281092	67466365	61784955	56250282
195	**196**	**197**	**198**	**199**	**200**
50873988	45675350	40669453	35869955	31296035	26962290
201	**202**	**203**	**204**	**205**	**206**
22884274	19082025	15571185	12367107	9491402	6960169
207	**208**	**209**	**210**	**211**	**212**
4789812	3002758	1616310	647264	118855	274

Table B.2: Statistics on the summations of square roots of $(n, k) = (1000, 4)$.

4	**5**	**6**	**7**	**8**	**9**
4	17	50	128	299	651
10	**11**	**12**	**13**	**14**	**15**
1312	2497	4528	7919	13259	21464
16	**17**	**18**	**19**	**20**	**21**

Continued on Next Page...

Table B.2 – Continued

33766	51722	77403	113364	162925	229953
22	**23**	**24**	**25**	**26**	**27**
319399	437431	590709	788355	1039928	1357857
28	**29**	**30**	**31**	**32**	**33**
1755316	2248742	2856881	3599975	4503890	5595350
34	**35**	**36**	**37**	**38**	**39**
6905693	8470771	10329432	12527095	15110538	18133144
40	**41**	**42**	**43**	**44**	**45**
21651965	25728582	30427681	35819193	41974094	48966774
46	**47**	**48**	**49**	**50**	**51**
56874026	65776432	75749543	86874225	99225454	112883109
52	**53**	**54**	**55**	**56**	**57**
127914469	144390021	162369319	181907566	203046425	225829072
58	**59**	**60**	**61**	**62**	**63**
250265896	276375109	304136746	333531976	364517079	397003873
64	**65**	**66**	**67**	**68**	**69**
430912753	466111103	502443997	539740555	577790727	616376654
70	**71**	**72**	**73**	**74**	**75**
655248449	694155665	732808268	770925737	808200617	844332516
76	**77**	**78**	**79**	**80**	**81**
878997048	911884053	942677958	971068074	996744522	1019421242
82	**83**	**84**	**85**	**86**	**87**

Continued on Next Page...

Table B.2 – Continued

1038815756	1054671487	1066745530	1074841243	1078773080	1078393131
88	**89**	**90**	**91**	**92**	**93**
1073600227	1064347442	1050620603	1032468820	1010015903	983420298
94	**95**	**96**	**97**	**98**	**99**
952951128	918927677	881770721	841910265	799745434	755720474
100	**101**	**102**	**103**	**104**	**105**
710233330	663697718	616513305	569088417	521801507	475027536
106	**107**	**108**	**109**	**110**	**111**
429126037	384439820	341281465	299958940	260730167	223850276
112	**113**	**114**	**115**	**116**	**117**
189529164	157951870	129256259	103557201	80910237	61340627
118	**119**	**120**	**121**	**122**	**123**
44809082	31246800	20506433	12402275	6669106	2991851
124	**125**	**126**			
974691	159884	2724			

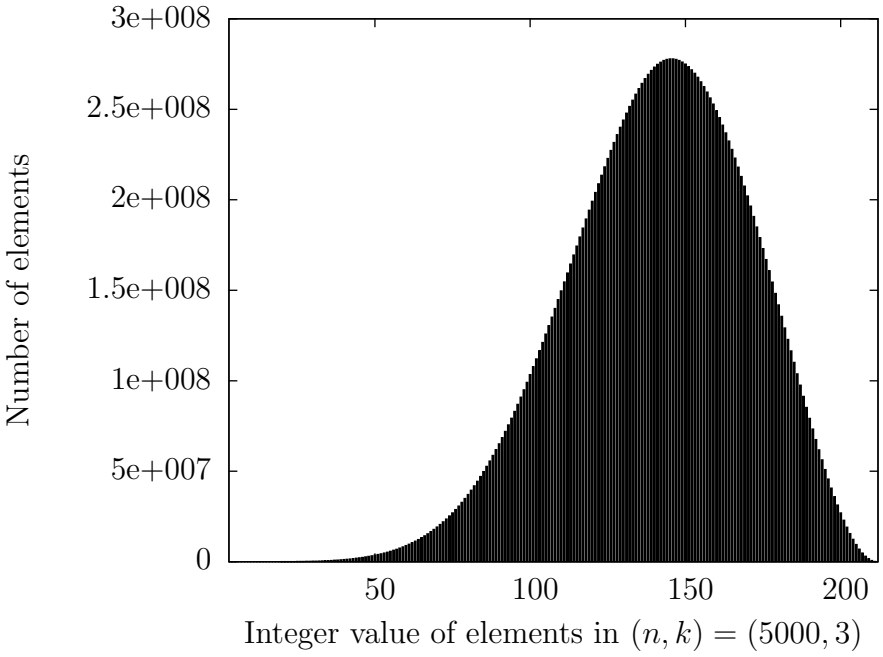

Figure B.1: Statistics on the summations of square roots of $(n, k) = (5000, 3)$.

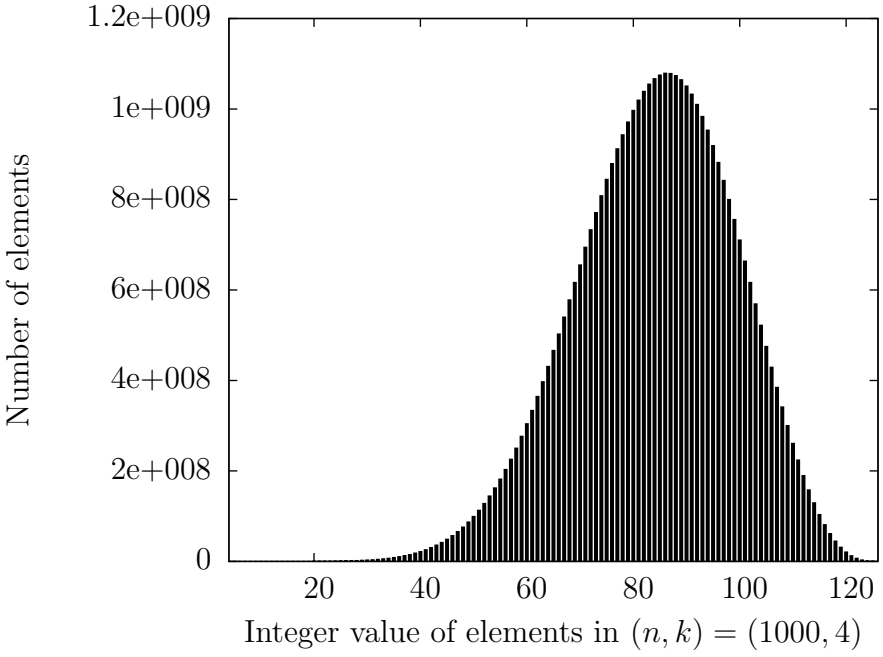

Figure B.2: Statistics on the summations of square roots of $(n, k) = (1000, 4)$.

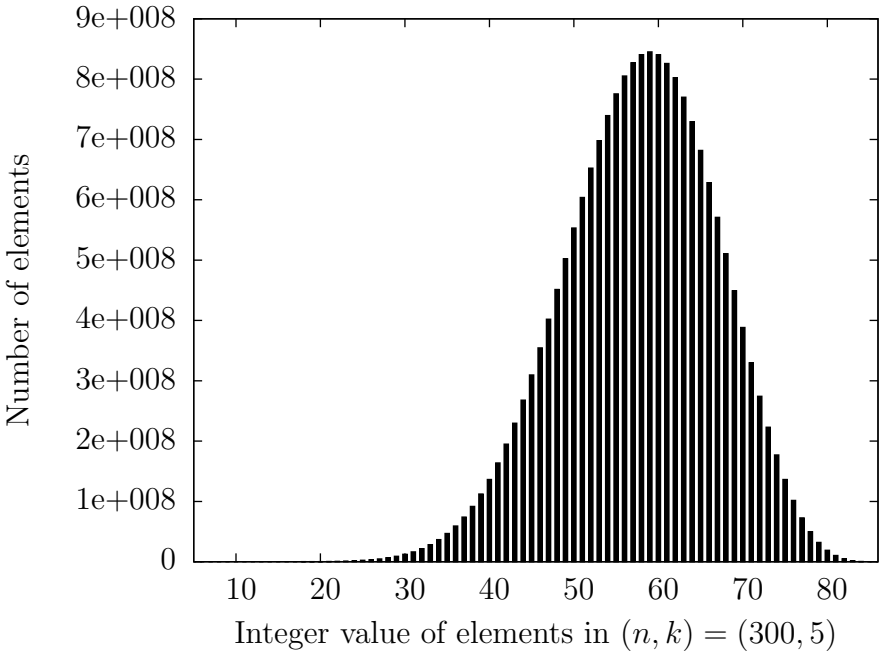

Figure B.3: Statistics on the summations of square roots of $(n, k) = (300, 5)$.

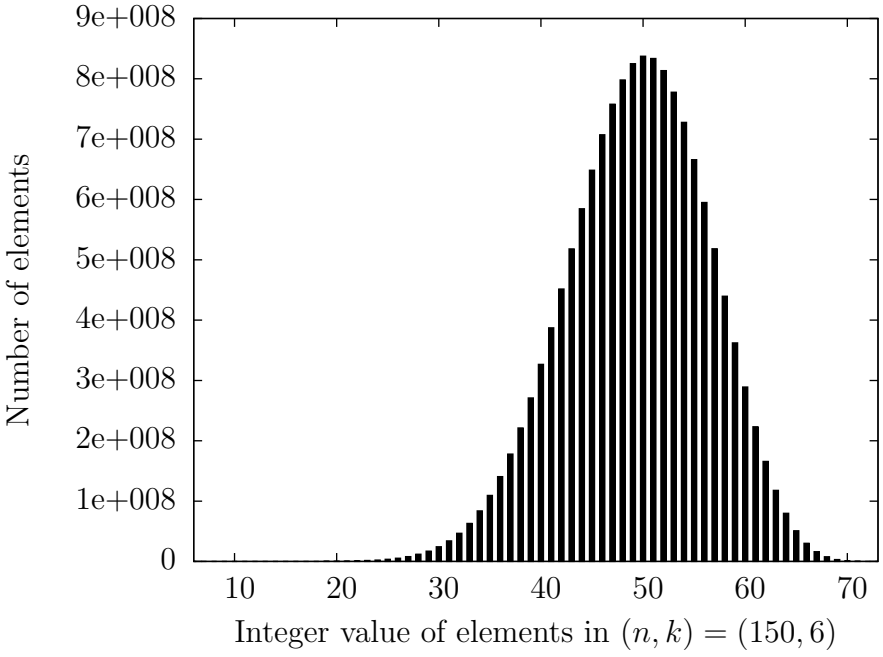

Figure B.4: Statistics on the summations of square roots of $(n, k) = (150, 6)$.

Table B.3: Statistics on the summations of square roots of $(n, k) = (300, 5)$.

5	**6**	**7**	**8**	**9**	**10**
4	17	54	146	355	817
11	**12**	**13**	**14**	**15**	**16**
1747	3549	6932	13073	23745	41809
17	**18**	**19**	**20**	**21**	**22**
71565	119269	194016	308499	480530	733937
23	**24**	**25**	**26**	**27**	**28**
1100836	1623688	2356719	3369470	4749103	6602766
29	**30**	**31**	**32**	**33**	**34**
9058272	12269794	16415340	21700160	28355408	36634084
35	**36**	**37**	**38**	**39**	**40**
46813055	59183469	74036388	91671343	112357622	136343901
41	**42**	**43**	**44**	**45**	**46**
163812365	194898187	229633010	267956016	309680149	354493114
47	**48**	**49**	**50**	**51**	**52**
401940354	451412562	502155395	553269137	603734631	652410302
53	**54**	**55**	**56**	**57**	**58**
698094835	739556412	775598887	805078395	827018698	840576595
59	**60**	**61**	**62**	**63**	**64**
845142458	840326357	826021999	802413863	769953678	729412528
65	**66**	**67**	**68**	**69**	**70**
681820418	628458668	570801455	510462944	449123074	388382164
71	**72**	**73**	**74**	**75**	**76**
329687126	274253086	223086114	176942554	136361965	101635242
77	**78**	**79**	**80**	**81**	**82**
72811905	49723807	31996843	19082226	10287515	4812225
83	**84**	**85**	**86**		
1816812	481367	61516	1125		

Table B.4: Statistics on the summations of square roots of $(n, k) = (150, 6)$.

6	**7**	**8**	**9**	**10**	**11**
4	17	56	156	394	930
12	**13**	**14**	**15**	**16**	**17**
2045	4305	8697	17029	32243	59483
18	**19**	**20**	**21**	**22**	**23**
106987	187776	322385	541654	891451	1437954
24	**25**	**26**	**27**	**28**	**29**
2275108	3532700	5383936	8059262	11850679	17124305
30	**31**	**32**	**33**	**34**	**35**
24322842	33964165	46637761	62986115	83670416	109341655
36	**37**	**38**	**39**	**40**	**41**
140579352	177833442	221344374	271077265	326648169	387246356
42	**43**	**44**	**45**	**46**	**47**
451635042	518106172	584548542	648484050	707219063	757978808
48	**49**	**50**	**51**	**52**	**53**
798083080	825164812	837394715	833636309	813582877	777834956
54	**55**	**56**	**57**	**58**	**59**
727868809	665948167	594995754	518364757	439599403	362169922
60	**61**	**62**	**63**	**64**	**65**
289178310	223124297	165719338	117861415	79694555	50744570
66	**67**	**68**	**69**	**70**	**71**
30004211	16155869	7684090	3076705	952216	194798
72	**73**				
17067	169				

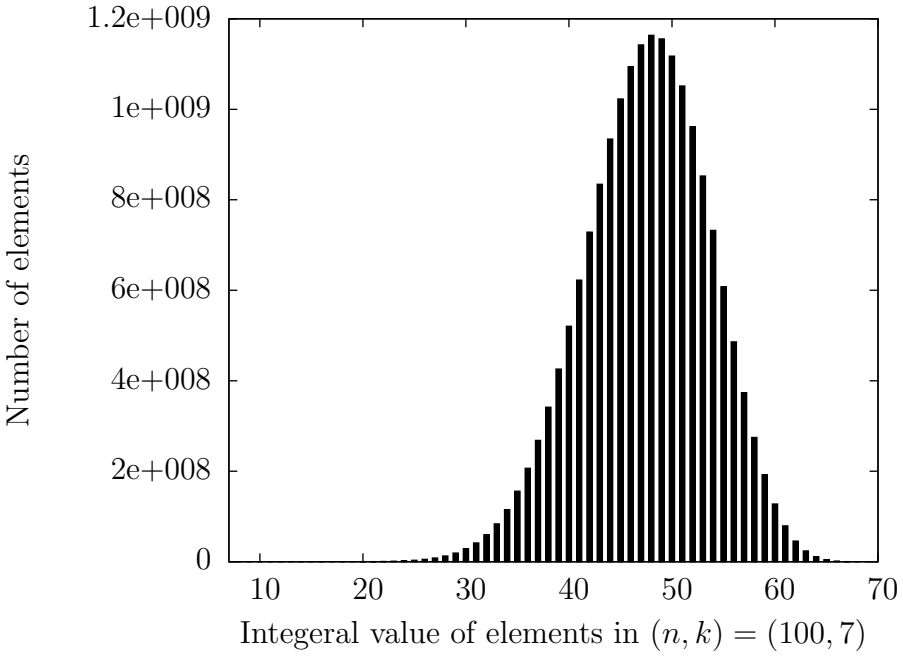

Figure B.5: Statistics on the summations of square roots of $(n, k) = (100, 7)$.

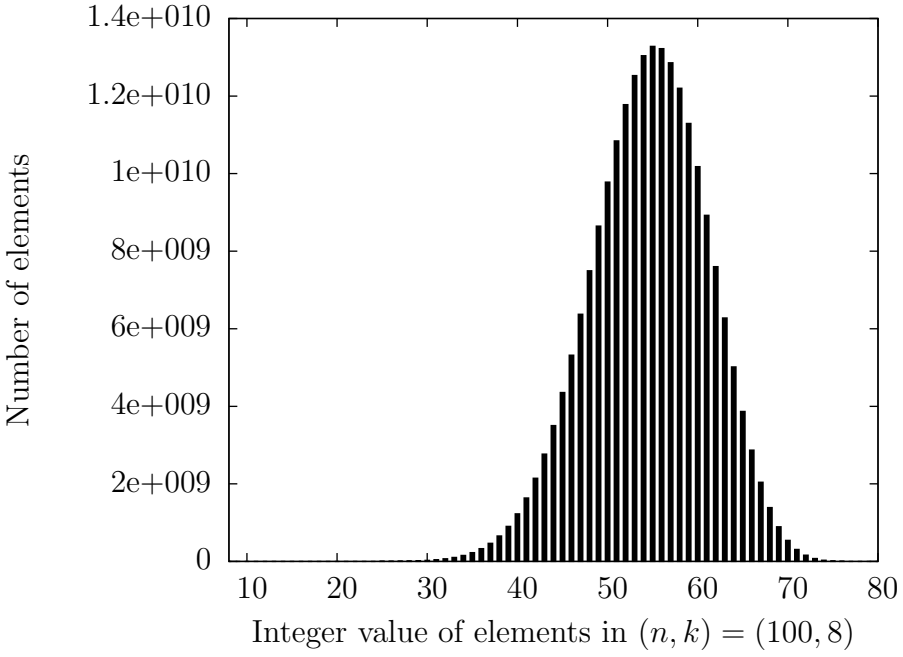

Figure B.6: Statistics on the summations of square roots of $(n, k) = (100, 8)$.

Table B.5: Statistics on the summations of square roots of $(n, k) = (100, 7)$.

7	**8**	**9**	**10**	**11**	**12**
4	17	57	161	418	1003
13	**14**	**15**	**16**	**17**	**18**
2259	4865	10044	20061	38742	72903
19	**20**	**21**	**22**	**23**	**24**
133706	239593	420279	722739	1218852	2017818
25	**26**	**27**	**28**	**29**	**30**
3280805	5239096	8218857	12664315	19165803	28482325
31	**32**	**33**	**34**	**35**	**36**
41554376	59503519	83607939	115241837	155784865	206478894
37	**38**	**39**	**40**	**41**	**42**
268254403	341520055	425961992	520334126	622307266	728445926
43	**44**	**45**	**46**	**47**	**48**
834229563	934295227	1022797808	1093860379	1142175328	1163570911
49	**50**	**51**	**52**	**53**	**54**
1155526520	1117588507	1051539385	961294902	852549403	732208073
55	**56**	**57**	**58**	**59**	**60**
607649679	486014737	373475729	274666260	192383944	127511613
61	**62**	**63**	**64**	**65**	**66**
79264404	45637971	23914891	11119037	4410314	1398655
67	**68**	**69**	**70**		
316043	40172	1476	1		

Table B.6: Statistics on the summations of square roots of $(n, k) = (100, 8)$.

8	**9**	**10**	**11**	**12**	**13**
4	17	57	164	431	1050
14	**15**	**16**	**17**	**18**	**19**
2405	5263	11049	22430	44087	84363
20	**21**	**22**	**23**	**24**	**25**
157463	287193	513089	899487	1548871	2622669
26	**27**	**28**	**29**	**30**	**31**
4371856	7178672	11618227	18538681	29172649	45274942
32	**33**	**34**	**35**	**36**	**37**
69296337	104581534	155593264	228140602	329565423	468876526
38	**39**	**40**	**41**	**42**	**43**
656736782	905290595	1227697273	1637381646	2146936750	2766660621
44	**45**	**46**	**47**	**48**	**49**
3502860953	4355940633	5318595539	6374248495	7496113100	8647060506
50	**51**	**52**	**53**	**54**	**55**
9780641949	10843296358	11777790745	12527657835	13042255212	13281868775
56	**57**	**58**	**59**	**60**	**61**
13222206419	12857766456	12203244922	11293200030	10179167253	8924982672
62	**63**	**64**	**65**	**66**	**67**
7600809547	6276409090	5014916390	3867671700	2870702945	2043201394
68	**69**	**70**	**71**	**72**	**73**
1388255200	895290965	543862936	308050514	160437835	75360655
74	**75**	**76**	**77**	**78**	**79**
31047417	10752045	2920890	551804	57323	1664
80					
1					

Appendix C

Statistics on Gaps

Table C.1: Statistics on the gaps of $(n, k) = (5000, 3)$.

$\mathbf{10^{-20} \sim 10^{-19}}$	$\mathbf{10^{-19} \sim 10^{-18}}$	$\mathbf{10^{-18} \sim 10^{-17}}$	$\mathbf{10^{-17} \sim 10^{-16}}$	$\mathbf{10^{-16} \sim 10^{-15}}$
4	2	30	402	3776
$\mathbf{10^{-15} \sim 10^{-14}}$	$\mathbf{10^{-14} \sim 10^{-13}}$	$\mathbf{10^{-13} \sim 10^{-12}}$	$\mathbf{10^{-12} \sim 10^{-11}}$	$\mathbf{10^{-11} \sim 10^{-10}}$
37714	382372	3786457	37907236	374151076
$\mathbf{10^{-10} \sim 10^{-9}}$	$\mathbf{10^{-9} \sim 10^{-8}}$	$\mathbf{10^{-8} \sim 10^{-7}}$	$\mathbf{10^{-7} \sim 10^{-6}}$	$\mathbf{10^{-6} \sim 10^{-5}}$
3342025737	13224270886	3671462146	138020004	7548734
$\mathbf{10^{-5} \sim 10^{-4}}$	$\mathbf{10^{-4} \sim 10^{-3}}$	$\mathbf{10^{-3} \sim 10^{-2}}$	$\mathbf{10^{-2} \sim 10^{-1}}$	$\mathbf{10^{-1} \sim 1}$
431036	27163	1921	103	5

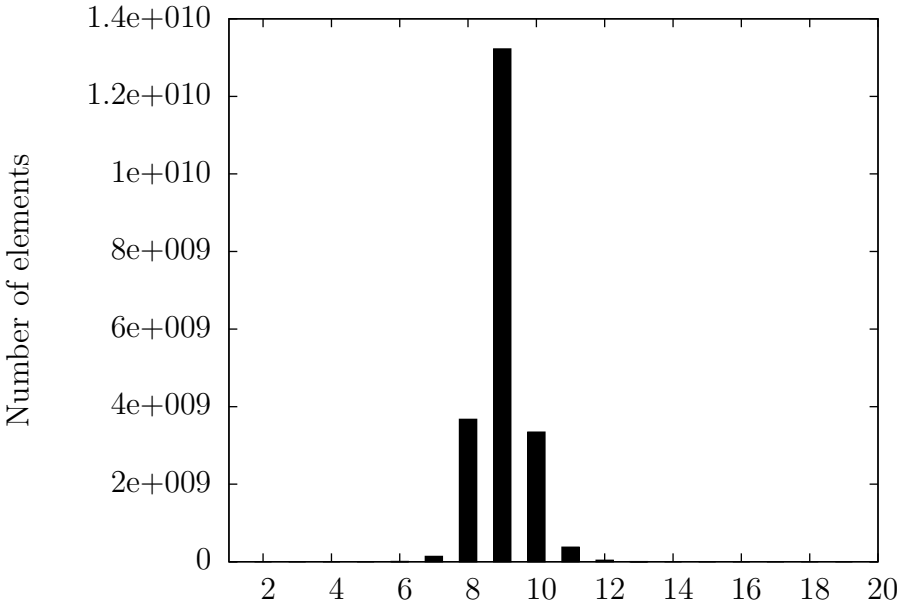

Figure C.1: Statistics on the gaps of $(n, k) = (5000, 3)$.

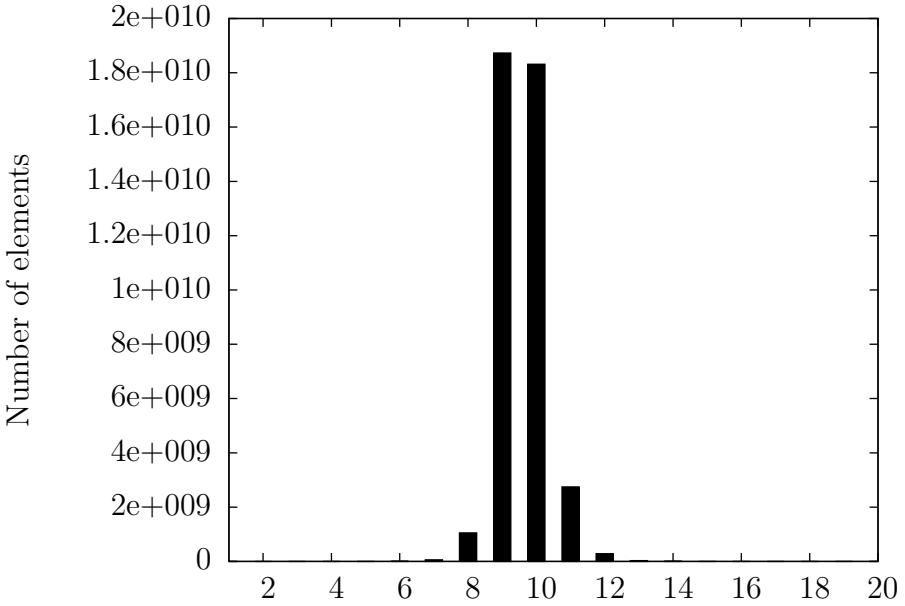

Figure C.2: Statistics on the gaps of $(n, k) = (1000, 4)$.

Table C.2: Statistics on the gaps of $(n, k) = (1000, 4)$.

$\mathbf{10^{-20} \sim 10^{-19}}$	$\mathbf{10^{-19} \sim 10^{-18}}$	$\mathbf{10^{-18} \sim 10^{-17}}$	$\mathbf{10^{-17} \sim 10^{-16}}$	$\mathbf{10^{-16} \sim 10^{-15}}$
1	23	333	2833	28173
$\mathbf{10^{-15} \sim 10^{-14}}$	$\mathbf{10^{-14} \sim 10^{-13}}$	$\mathbf{10^{-13} \sim 10^{-12}}$	$\mathbf{10^{-12} \sim 10^{-11}}$	$\mathbf{10^{-11} \sim 10^{-10}}$
289942	2879619	28837669	287029389	2747043792
$\mathbf{10^{-10} \sim 10^{-9}}$	$\mathbf{10^{-9} \sim 10^{-8}}$	$\mathbf{10^{-8} \sim 10^{-7}}$	$\mathbf{10^{-7} \sim 10^{-6}}$	$\mathbf{10^{-6} \sim 10^{-5}}$
18317656395	18727827263	1051997943	58540923	3769056
$\mathbf{10^{-5} \sim 10^{-4}}$	$\mathbf{10^{-4} \sim 10^{-3}}$	$\mathbf{10^{-3} \sim 10^{-2}}$	$\mathbf{10^{-2} \sim 10^{-1}}$	$\mathbf{10^{-1} \sim 1}$
243811	17596	1383	125	5

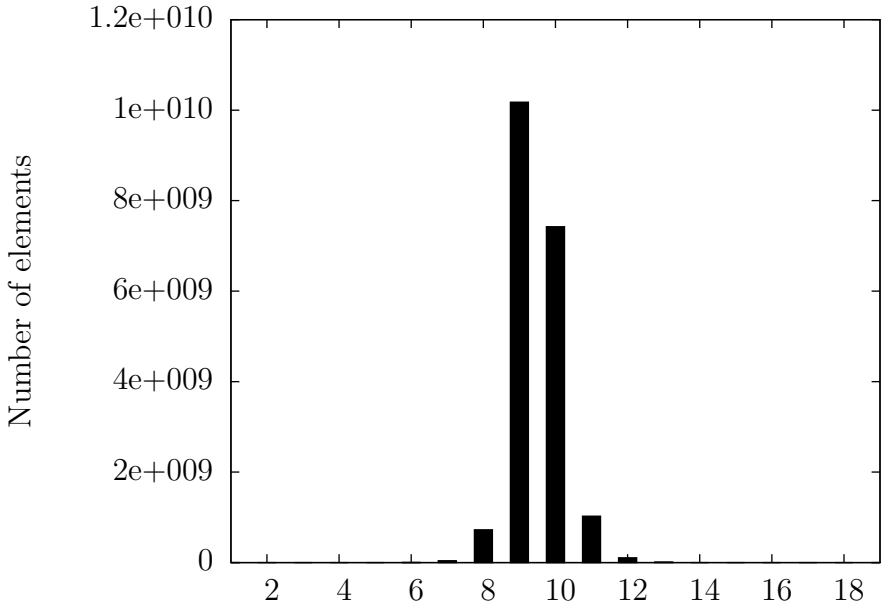

Figure C.3: Statistics on the gaps of $(n, k) = (300, 5)$.

Table C.3: Statistics on the gaps of $(n, k) = (300, 5)$.

$\mathbf{10^{-19} \sim 10^{-18}}$	$\mathbf{10^{-18} \sim 10^{-17}}$	$\mathbf{10^{-17} \sim 10^{-16}}$	$\mathbf{10^{-16} \sim 10^{-15}}$	$\mathbf{10^{-15} \sim 10^{-14}}$
9	70	1013	10708	106142
$\mathbf{10^{-14} \sim 10^{-13}}$	$\mathbf{10^{-13} \sim 10^{-12}}$	$\mathbf{10^{-12} \sim 10^{-11}}$	$\mathbf{10^{-11} \sim 10^{-10}}$	$\mathbf{10^{-10} \sim 10^{-9}}$
1057841	10703286	106097453	1026867788	7424234818
$\mathbf{10^{-9} \sim 10^{-8}}$	$\mathbf{10^{-8} \sim 10^{-7}}$	$\mathbf{10^{-7} \sim 10^{-6}}$	$\mathbf{10^{-6} \sim 10^{-5}}$	$\mathbf{10^{-5} \sim 10^{-4}}$
10175945086	724055328	41202961	2823186	199757
$\mathbf{10^{-4} \sim 10^{-3}}$	$\mathbf{10^{-3} \sim 10^{-2}}$	$\mathbf{10^{-2} \sim 10^{-1}}$	$\mathbf{10^{-1} \sim 1}$	
14671	1338	99	5	

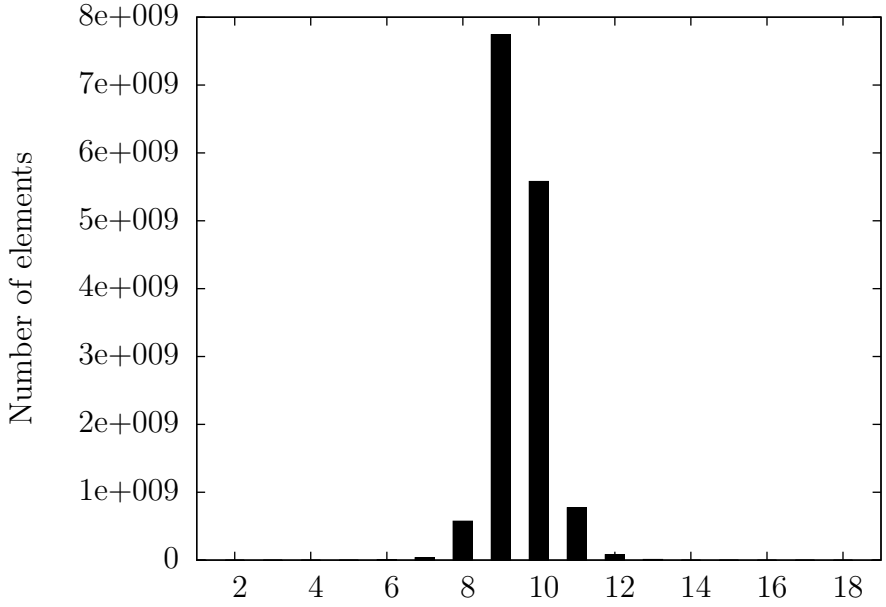

Figure C.4: Statistics on the gaps of $(n, k) = (150, 6)$.

Table C.4: Statistics on the gaps of $(n, k) = (150, 6)$.

$\mathbf{10^{-19} \sim 10^{-18}}$	$\mathbf{10^{-18} \sim 10^{-17}}$	$\mathbf{10^{-17} \sim 10^{-16}}$	$\mathbf{10^{-16} \sim 10^{-15}}$	$\mathbf{10^{-15} \sim 10^{-14}}$
3	58	530	7330	72503
$\mathbf{10^{-14} \sim 10^{-13}}$	$\mathbf{10^{-13} \sim 10^{-12}}$	$\mathbf{10^{-12} \sim 10^{-11}}$	$\mathbf{10^{-11} \sim 10^{-10}}$	$\mathbf{10^{-10} \sim 10^{-9}}$
799716	7907620	79436984	770641372	5576744694
$\mathbf{10^{-9} \sim 10^{-8}}$	$\mathbf{10^{-8} \sim 10^{-7}}$	$\mathbf{10^{-7} \sim 10^{-6}}$	$\mathbf{10^{-6} \sim 10^{-5}}$	$\mathbf{10^{-5} \sim 10^{-4}}$
7741146745	570941357	34146678	2469429	182632
$\mathbf{10^{-4} \sim 10^{-3}}$	$\mathbf{10^{-3} \sim 10^{-2}}$	$\mathbf{10^{-2} \sim 10^{-1}}$	$\mathbf{10^{-1} \sim 1}$	
15269	1304	86	5	

Table C.5: Statistics on the gaps of $(n, k) = (100, 7)$.

$\mathbf{10^{-19} \sim 10^{-18}}$	$\mathbf{10^{-18} \sim 10^{-17}}$	$\mathbf{10^{-17} \sim 10^{-16}}$	$\mathbf{10^{-16} \sim 10^{-15}}$	$\mathbf{10^{-15} \sim 10^{-14}}$
7	47	1245	14139	129248
$\mathbf{10^{-14} \sim 10^{-13}}$	$\mathbf{10^{-13} \sim 10^{-12}}$	$\mathbf{10^{-12} \sim 10^{-11}}$	$\mathbf{10^{-11} \sim 10^{-10}}$	$\mathbf{10^{-10} \sim 10^{-9}}$
1459473	13100265	132767395	1272832428	8256755966
$\mathbf{10^{-9} \sim 10^{-8}}$	$\mathbf{10^{-8} \sim 10^{-7}}$	$\mathbf{10^{-7} \sim 10^{-6}}$	$\mathbf{10^{-6} \sim 10^{-5}}$	$\mathbf{10^{-5} \sim 10^{-4}}$
7766837445	463570895	30415764	2314151	176109
$\mathbf{10^{-4} \sim 10^{-3}}$	$\mathbf{10^{-3} \sim 10^{-2}}$	$\mathbf{10^{-2} \sim 10^{-1}}$	$\mathbf{10^{-1} \sim 1}$	
14890	1300	80	5	

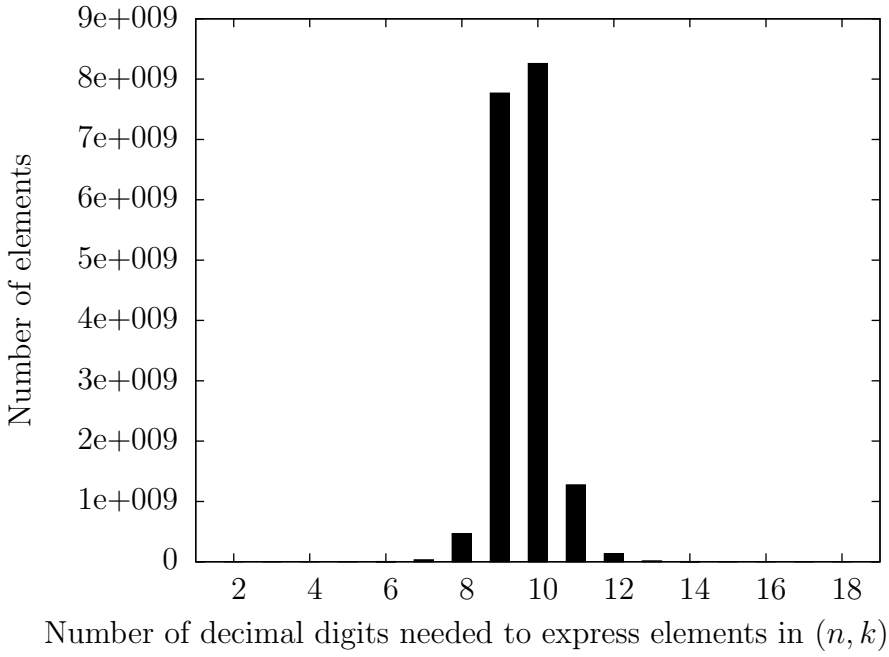

Figure C.5: Statistics on the gaps of $(n, k) = (100, 7)$.

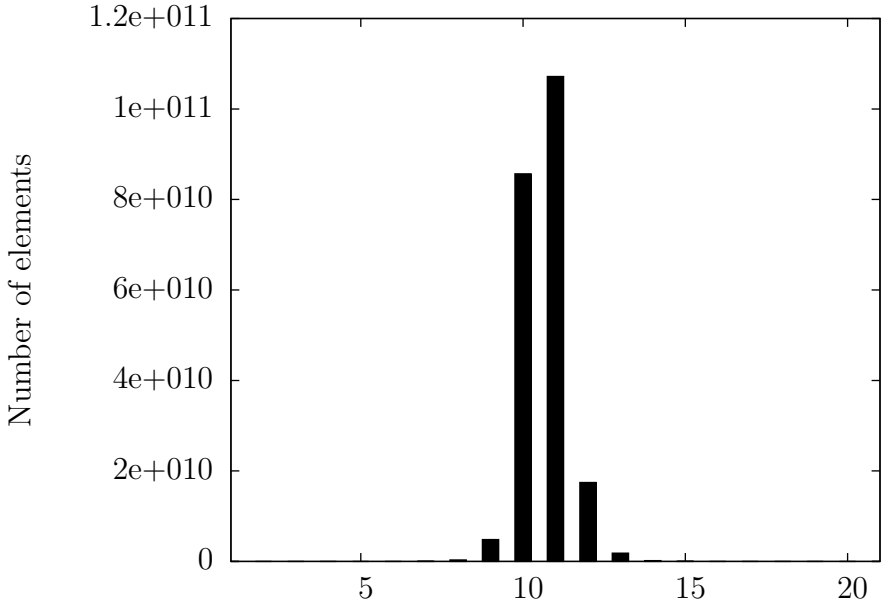

Figure C.6: Statistics on the gaps of $(n, k) = (100, 8)$.

Table C.6: Statistics on the gaps of $(n, k) = (100, 8)$.

$\mathbf{10^{-21} \sim 10^{-20}}$	$\mathbf{10^{-20} \sim 10^{-19}}$	$\mathbf{10^{-19} \sim 10^{-18}}$	$\mathbf{10^{-18} \sim 10^{-17}}$	$\mathbf{10^{-17} \sim 10^{-16}}$
4	118	1598	12961	194745
$\mathbf{10^{-16} \sim 10^{-15}}$	$\mathbf{10^{-15} \sim 10^{-14}}$	$\mathbf{10^{-14} \sim 10^{-13}}$	$\mathbf{10^{-13} \sim 10^{-12}}$	$\mathbf{10^{-12} \sim 10^{-11}}$
1899601	18367740	188014906	1832870080	17451973275
$\mathbf{10^{-11} \sim 10^{-10}}$	$\mathbf{10^{-10} \sim 10^{-9}}$	$\mathbf{10^{-9} \sim 10^{-8}}$	$\mathbf{10^{-8} \sim 10^{-7}}$	$\mathbf{10^{-7} \sim 10^{-6}}$
107179609158	85672158135	4834929115	330274907	25729746
$\mathbf{10^{-6} \sim 10^{-5}}$	$\mathbf{10^{-5} \sim 10^{-4}}$	$\mathbf{10^{-4} \sim 10^{-3}}$	$\mathbf{10^{-3} \sim 10^{-2}}$	$\mathbf{10^{-2} \sim 10^{-1}}$
2095578	163106	14501	1279	80
$\mathbf{10^{-1} \sim 1}$				
5				

Appendix D

Required Minimum Precision

Table D.1: The smallest difference of $k = 3$.

k	n	r(n,k)	Sum of Square Roots
3	3153...5000	2.84×10^{-20}	$(\sqrt{29} + \sqrt{1097} + \sqrt{3153})$ - $(\sqrt{226} + \sqrt{987} + \sqrt{2324})$
3	2025...3152	6.80×10^{-20}	$(\sqrt{190} + \sqrt{398} + \sqrt{1482})$ - $(\sqrt{176} + \sqrt{195} + \sqrt{2025})$
3	1738...2024	8.84×10^{-17}	$(\sqrt{831} + \sqrt{905} + \sqrt{1738})$ - $(\sqrt{511} + \sqrt{1384} + \sqrt{1664})$
3	1720...1737	2.36×10^{-16}	$(\sqrt{924} + \sqrt{1336} + \sqrt{1517})$ - $(\sqrt{629} + \sqrt{1548} + \sqrt{1720})$
3	1489...1719	2.82×10^{-16}	$(\sqrt{724} + \sqrt{1125} + \sqrt{1420})$ - $(\sqrt{645} + \sqrt{1166} + \sqrt{1489})$
3	1019...1488	7.64×10^{-16}	$(\sqrt{32} + \sqrt{951} + \sqrt{1019})$ - $(\sqrt{255} + \sqrt{443} + \sqrt{986})$
3	500...1018	4.12×10^{-15}	$(\sqrt{105} + \sqrt{287} + \sqrt{484})$ - $(\sqrt{96} + \sqrt{290} + \sqrt{500})$
3	472...499	8.54×10^{-14}	$(\sqrt{41} + \sqrt{247} + \sqrt{472})$ - $(\sqrt{76} + \sqrt{274} + \sqrt{345})$
3	276...471	3.14×10^{-13}	$(\sqrt{10} + \sqrt{234} + \sqrt{276})$ - $(\sqrt{49} + \sqrt{178} + \sqrt{217})$
3	222...275	4.81×10^{-13}	$(\sqrt{90} + \sqrt{99} + \sqrt{222})$ - $(\sqrt{85} + \sqrt{110} + \sqrt{214})$
3	211...221	8.22×10^{-12}	$(\sqrt{33} + \sqrt{61} + \sqrt{211})$ - $(\sqrt{53} + \sqrt{90} + \sqrt{128})$
3	207...210	2.45×10^{-11}	$(\sqrt{67} + \sqrt{154} + \sqrt{189})$ - $(\sqrt{31} + \sqrt{207} + \sqrt{207})$
3	182...206	3.62×10^{-11}	$(\sqrt{25} + \sqrt{135} + \sqrt{182})$ - $(\sqrt{58} + \sqrt{111} + \sqrt{143})$
3	158...181	5.75×10^{-11}	$(\sqrt{4} + \sqrt{152} + \sqrt{158})$ - $(\sqrt{15} + \sqrt{111} + \sqrt{156})$
3	115...157	6.86×10^{-11}	$(\sqrt{38} + \sqrt{43} + \sqrt{105})$ - $(\sqrt{36} + \sqrt{39} + \sqrt{115})$

Continued on Next Page...

Table D.1 – Continued

k	n	r(n,k)	Sum of Square Roots
3	103...114	4.86×10^{-10}	$(\sqrt{56}+\sqrt{66}+\sqrt{96})$ - $(\sqrt{50}+\sqrt{67}+\sqrt{103})$
3	98...102	8.45×10^{-10}	$(\sqrt{31}+\sqrt{48}+\sqrt{98})$ - $(\sqrt{42}+\sqrt{42}+\sqrt{89})$
3	60...97	1.55×10^{-09}	$(\sqrt{12}+\sqrt{17}+\sqrt{56})$ - $(\sqrt{1}+\sqrt{40}+\sqrt{60})$
3	52...59	2.21×10^{-08}	$(\sqrt{10}+\sqrt{36}+\sqrt{45})$ - $(\sqrt{8}+\sqrt{34}+\sqrt{52})$
3	39...51	1.27×10^{-07}	$(\sqrt{9}+\sqrt{33}+\sqrt{35})$ - $(\sqrt{11}+\sqrt{26}+\sqrt{39})$
3	38	3.34×10^{-07}	$(\sqrt{9}+\sqrt{24}+\sqrt{38})$ - $(\sqrt{5}+\sqrt{33}+\sqrt{37})$
3	21...37	4.00×10^{-07}	$(\sqrt{6}+\sqrt{21}+\sqrt{21})$ - $(\sqrt{14}+\sqrt{15}+\sqrt{16})$
3	20	1.26×10^{-05}	$(\sqrt{5}+\sqrt{14}+\sqrt{20})$ - $(\sqrt{3}+\sqrt{19}+\sqrt{19})$
3	15...19	4.69×10^{-05}	$(\sqrt{4}+\sqrt{13}+\sqrt{14})$ - $(\sqrt{7}+\sqrt{8}+\sqrt{15})$
3	13, 14	1.49×10^{-04}	$(\sqrt{1}+\sqrt{13}+\sqrt{13})$ - $(\sqrt{3}+\sqrt{10}+\sqrt{11})$
3	11, 12	1.94×10^{-04}	$(\sqrt{2}+\sqrt{10}+\sqrt{11})$ - $(\sqrt{5}+\sqrt{8}+\sqrt{8})$
3	7...10	2.04×10^{-03}	$(\sqrt{2}+\sqrt{2}+\sqrt{7})$ - $(\sqrt{1}+\sqrt{5}+\sqrt{5})$
3	5, 6	6.57×10^{-03}	$(\sqrt{2}+\sqrt{2}+\sqrt{2})$ - $(\sqrt{1}+\sqrt{1}+\sqrt{5})$
3	4	9.64×10^{-02}	$(\sqrt{2}+\sqrt{2}+\sqrt{4})$ - $(\sqrt{1}+\sqrt{3}+\sqrt{4})$
3	3	9.64×10^{-02}	$(\sqrt{1}+\sqrt{2}+\sqrt{2})$ - $(\sqrt{1}+\sqrt{1}+\sqrt{3})$
3	2	4.14×10^{-01}	$(\sqrt{1}+\sqrt{1}+\sqrt{2})$ - $(\sqrt{1}+\sqrt{1}+\sqrt{1})$

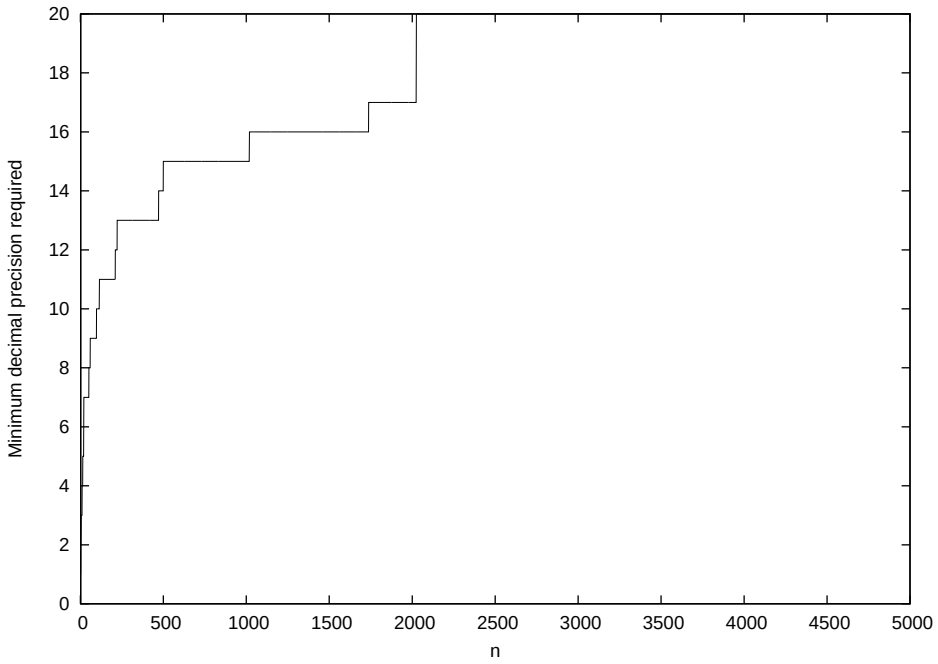

Figure D.1: Minimum precision required when n varies and $k = 3$.

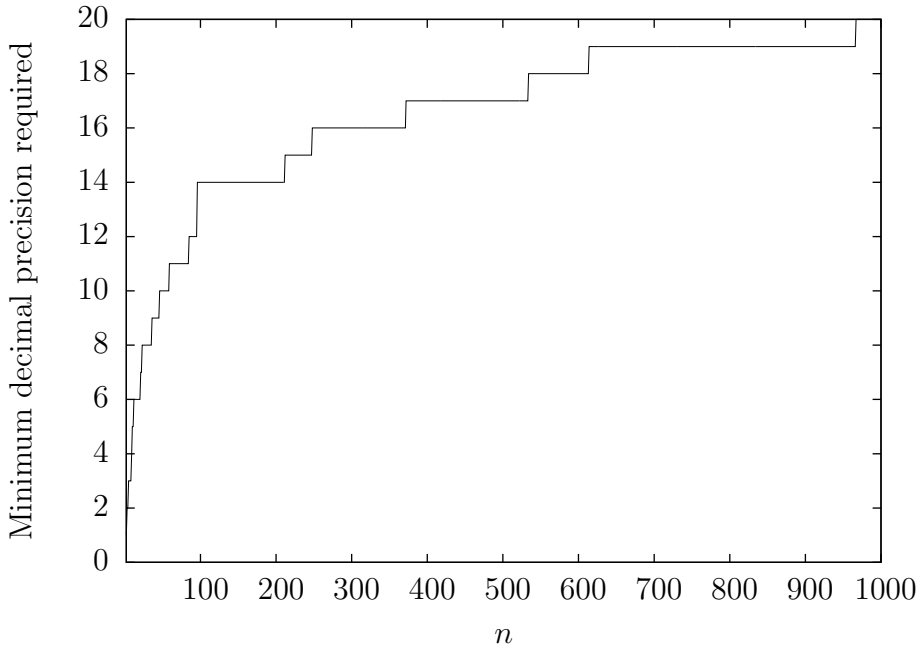

Figure D.2: Minimum precision required when n varies and $k = 4$.

Table D.2: The smallest difference of $k = 4$.

k	**n**	**r(n,k)**	**Sum of Square Roots**
4	967...1000	9.15×10^{-20}	$(\sqrt{154} + \sqrt{381} + \sqrt{770} + \sqrt{774})$ - $(\sqrt{128} + \sqrt{394} + \sqrt{637} + \sqrt{967})$
4	685...966	1.17×10^{-19}	$(\sqrt{73} + \sqrt{630} + \sqrt{640} + \sqrt{685})$ - $(\sqrt{143} + \sqrt{564} + \sqrt{569} + \sqrt{653})$
4	614...684	8.14×10^{-19}	$(\sqrt{116} + \sqrt{208} + \sqrt{329} + \sqrt{443})$ - $(\sqrt{50} + \sqrt{95} + \sqrt{519} + \sqrt{614})$
4	574...613	3.83×10^{-18}	$(\sqrt{214} + \sqrt{218} + \sqrt{223} + \sqrt{310})$ - $(\sqrt{4} + \sqrt{300} + \sqrt{348} + \sqrt{574})$
4	534...573	7.07×10^{-18}	$(\sqrt{5} + \sqrt{54} + \sqrt{519} + \sqrt{534})$ - $(\sqrt{95} + \sqrt{189} + \sqrt{197} + \sqrt{322})$
4	503...533	1.73×10^{-17}	$(\sqrt{174} + \sqrt{264} + \sqrt{311} + \sqrt{503})$ - $(\sqrt{104} + \sqrt{372} + \sqrt{384} + \sqrt{417})$
4	469...502	2.07×10^{-17}	$(\sqrt{79} + \sqrt{324} + \sqrt{373} + \sqrt{446})$ - $(\sqrt{165} + \sqrt{234} + \sqrt{307} + \sqrt{469})$
4	421...468	3.46×10^{-17}	$(\sqrt{63} + \sqrt{253} + \sqrt{357} + \sqrt{380})$ - $(\sqrt{21} + \sqrt{285} + \sqrt{410} + \sqrt{421})$
4	413...420	5.32×10^{-17}	$(\sqrt{10} + \sqrt{219} + \sqrt{383} + \sqrt{412})$ - $(\sqrt{1} + \sqrt{313} + \sqrt{354} + \sqrt{413})$
4	372...412	8.53×10^{-17}	$(\sqrt{130} + \sqrt{206} + \sqrt{208} + \sqrt{273})$ - $(\sqrt{73} + \sqrt{197} + \sqrt{220} + \sqrt{372})$
4	344...371	2.38×10^{-16}	$(\sqrt{131} + \sqrt{158} + \sqrt{171} + \sqrt{336})$

Continued on Next Page...

Table D.2 – Continued

k	n	r(n,k)	Sum of Square Roots
			- $(\sqrt{82}+\sqrt{156}+\sqrt{235}+\sqrt{344})$
4	248...343	6.92×10^{-16}	$(\sqrt{50}+\sqrt{205}+\sqrt{212}+\sqrt{240})$ - $(\sqrt{48}+\sqrt{196}+\sqrt{218}+\sqrt{248})$
4	212...247	5.43×10^{-15}	$(\sqrt{148}+\sqrt{167}+\sqrt{182}+\sqrt{212})$ - $(\sqrt{118}+\sqrt{190}+\sqrt{197}+\sqrt{209})$
4	194...211	1.55×10^{-14}	$(\sqrt{39}+\sqrt{141}+\sqrt{167}+\sqrt{185})$ - $(\sqrt{46}+\sqrt{103}+\sqrt{190}+\sqrt{194})$
4	152...193	3.43×10^{-14}	$(\sqrt{34}+\sqrt{56}+\sqrt{151}+\sqrt{152})$ - $(\sqrt{64}+\sqrt{67}+\sqrt{91}+\sqrt{149})$
4	96...151	5.04×10^{-14}	$(\sqrt{45}+\sqrt{63}+\sqrt{91}+\sqrt{96})$ - $(\sqrt{44}+\sqrt{65}+\sqrt{93}+\sqrt{93})$
4	89...95	3.60×10^{-12}	$(\sqrt{18}+\sqrt{38}+\sqrt{62}+\sqrt{84})$ - $(\sqrt{1}+\sqrt{66}+\sqrt{79}+\sqrt{89})$
4	85...88	9.06×10^{-12}	$(\sqrt{7}+\sqrt{15}+\sqrt{51}+\sqrt{62})$ - $(\sqrt{3}+\sqrt{21}+\sqrt{36}+\sqrt{85})$
4	59...84	2.97×10^{-11}	$(\sqrt{20}+\sqrt{33}+\sqrt{33}+\sqrt{53})$ - $(\sqrt{13}+\sqrt{26}+\sqrt{47}+\sqrt{59})$
4	46...58	1.85×10^{-10}	$(\sqrt{16}+\sqrt{34}+\sqrt{41}+\sqrt{46})$ - $(\sqrt{24}+\sqrt{30}+\sqrt{37}+\sqrt{43})$
4	36...45	6.91×10^{-09}	$(\sqrt{23}+\sqrt{24}+\sqrt{34}+\sqrt{35})$ - $(\sqrt{21}+\sqrt{28}+\sqrt{31}+\sqrt{36})$
4	33...35	2.84×10^{-08}	$(\sqrt{8}+\sqrt{16}+\sqrt{20}+\sqrt{33})$

Continued on Next Page...

Table D.2 – Continued

k	n	r(n,k)	Sum of Square Roots
			- $(\sqrt{12}+\sqrt{17}+\sqrt{19}+\sqrt{26})$
4	31, 32	5.02×10^{-08}	$(\sqrt{2}+\sqrt{15}+\sqrt{24}+\sqrt{31})$ - $(\sqrt{9}+\sqrt{10}+\sqrt{23}+\sqrt{23})$
4	23...30	6.17×10^{-08}	$(\sqrt{2}+\sqrt{14}+\sqrt{18}+\sqrt{23})$ - $(\sqrt{6}+\sqrt{10}+\sqrt{16}+\sqrt{21})$
4	21, 22	4.00×10^{-07}	$(\sqrt{6}+\sqrt{10}+\sqrt{21}+\sqrt{21})$ - $(\sqrt{10}+\sqrt{14}+\sqrt{15}+\sqrt{16})$
4	19, 20	1.38×10^{-06}	$(\sqrt{7}+\sqrt{10}+\sqrt{12}+\sqrt{18})$ - $(\sqrt{2}+\sqrt{14}+\sqrt{16}+\sqrt{19})$
4	16...18	3.20×10^{-06}	$(\sqrt{6}+\sqrt{10}+\sqrt{11}+\sqrt{14})$ - $(\sqrt{5}+\sqrt{8}+\sqrt{13}+\sqrt{16})$
4	14, 15	4.25×10^{-06}	$(\sqrt{3}+\sqrt{8}+\sqrt{11}+\sqrt{13})$ - $(\sqrt{6}+\sqrt{7}+\sqrt{7}+\sqrt{14})$
4	12, 13	4.82×10^{-06}	$(\sqrt{3}+\sqrt{3}+\sqrt{4}+\sqrt{12})$ - $(\sqrt{2}+\sqrt{5}+\sqrt{6}+\sqrt{8})$
4	11	4.84×10^{-05}	$(\sqrt{1}+\sqrt{3}+\sqrt{9}+\sqrt{11})$ - $(\sqrt{2}+\sqrt{5}+\sqrt{5}+\sqrt{10})$
4	10	6.02×10^{-05}	$(\sqrt{3}+\sqrt{5}+\sqrt{5}+\sqrt{5})$ - $(\sqrt{2}+\sqrt{2}+\sqrt{6}+\sqrt{10})$
4	9	1.09×10^{-04}	$(\sqrt{1}+\sqrt{1}+\sqrt{7}+\sqrt{9})$ - $(\sqrt{3}+\sqrt{3}+\sqrt{3}+\sqrt{6})$
4	8	1.03×10^{-03}	$(\sqrt{3}+\sqrt{3}+\sqrt{3}+\sqrt{8})$

Continued on Next Page...

Table D.2 – Continued

k	n	r(n,k)	Sum of Square Roots
			- $(\sqrt{1}+\sqrt{3}+\sqrt{7}+\sqrt{7})$
4	7	2.04×10^{-03}	$(\sqrt{2}+\sqrt{2}+\sqrt{3}+\sqrt{7})$ - $(\sqrt{1}+\sqrt{3}+\sqrt{5}+\sqrt{5})$
4	5, 6	6.57×10^{-03}	$(\sqrt{1}+\sqrt{2}+\sqrt{2}+\sqrt{2})$ - $(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{5})$
4	4	7.52×10^{-02}	$(\sqrt{1}+\sqrt{1}+\sqrt{3}+\sqrt{4})$ - $(\sqrt{2}+\sqrt{2}+\sqrt{2}+\sqrt{2})$
4	3	9.64×10^{-02}	$(\sqrt{1}+\sqrt{1}+\sqrt{2}+\sqrt{2})$ - $(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{3})$
4	2	4.14×10^{-01}	$(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{2})$ - $(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1})$

Table D.3: The smallest difference of $k = 5$.

k	n	r(n,k)	Sum of Square Roots
5	269...300	1.45×10^{-19}	$(\sqrt{101}+\sqrt{131}+\sqrt{185}+\sqrt{211}+\sqrt{212})$ - $(\sqrt{61}+\sqrt{128}+\sqrt{154}+\sqrt{264}+\sqrt{269})$
5	264...268	1.64×10^{-19}	$(\sqrt{41}+\sqrt{64}+\sqrt{108}+\sqrt{156}+\sqrt{260})$ - $(\sqrt{70}+\sqrt{74}+\sqrt{97}+\sqrt{107}+\sqrt{264})$

Continued on Next Page...

Table D.3 – Continued

k	n	r(n,k)	Sum of Square Roots
5	190...263	1.85×10^{-19}	$(\sqrt{11} + \sqrt{67} + \sqrt{101} + \sqrt{127} + \sqrt{157})$ - $(\sqrt{41} + \sqrt{55} + \sqrt{57} + \sqrt{104} + \sqrt{190})$
5	179...189	1.12×10^{-17}	$(\sqrt{11} + \sqrt{63} + \sqrt{79} + \sqrt{141} + \sqrt{179})$ - $(\sqrt{7} + \sqrt{96} + \sqrt{104} + \sqrt{107} + \sqrt{154})$
5	158...178	1.66×10^{-17}	$(\sqrt{47} + \sqrt{50} + \sqrt{73} + \sqrt{103} + \sqrt{158})$ - $(\sqrt{13} + \sqrt{78} + \sqrt{89} + \sqrt{130} + \sqrt{142})$
5	148...157	2.99×10^{-16}	$(\sqrt{49} + \sqrt{95} + \sqrt{105} + \sqrt{133} + \sqrt{148})$ - $(\sqrt{52} + \sqrt{85} + \sqrt{119} + \sqrt{127} + \sqrt{146})$
5	143...147	4.80×10^{-16}	$(\sqrt{45} + \sqrt{67} + \sqrt{103} + \sqrt{106} + \sqrt{112})$ - $(\sqrt{43} + \sqrt{56} + \sqrt{89} + \sqrt{110} + \sqrt{143})$
5	117...142	5.08×10^{-16}	$(\sqrt{47} + \sqrt{60} + \sqrt{76} + \sqrt{92} + \sqrt{102})$ - $(\sqrt{30} + \sqrt{65} + \sqrt{86} + \sqrt{88} + \sqrt{117})$
5	112...116	4.97×10^{-15}	$(\sqrt{34} + \sqrt{46} + \sqrt{77} + \sqrt{92} + \sqrt{112})$ - $(\sqrt{14} + \sqrt{73} + \sqrt{87} + \sqrt{99} + \sqrt{100})$
5	94...111	5.66×10^{-15}	$(\sqrt{36} + \sqrt{40} + \sqrt{83} + \sqrt{86} + \sqrt{94})$ - $(\sqrt{52} + \sqrt{62} + \sqrt{66} + \sqrt{69} + \sqrt{79})$
5	83...93	1.16×10^{-14}	$(\sqrt{15} + \sqrt{16} + \sqrt{62} + \sqrt{67} + \sqrt{72})$ - $(\sqrt{3} + \sqrt{38} + \sqrt{54} + \sqrt{65} + \sqrt{83})$
5	81, 82	1.49×10^{-13}	$(\sqrt{17} + \sqrt{38} + \sqrt{65} + \sqrt{67} + \sqrt{77})$ - $(\sqrt{19} + \sqrt{35} + \sqrt{57} + \sqrt{72} + \sqrt{81})$
5	75...80	4.13×10^{-13}	$(\sqrt{14} + \sqrt{45} + \sqrt{64} + \sqrt{65} + \sqrt{66})$ - $(\sqrt{26} + \sqrt{30} + \sqrt{47} + \sqrt{73} + \sqrt{75})$

Continued on Next Page...

Table D.3 – Continued

k	n	r(n,k)	Sum of Square Roots
5	62...74	4.93×10^{-13}	$(\sqrt{2}+\sqrt{34}+\sqrt{45}+\sqrt{52}+\sqrt{61})$ - $(\sqrt{15}+\sqrt{21}+\sqrt{38}+\sqrt{42}+\sqrt{62})$
5	59...61	5.04×10^{-13}	$(\sqrt{7}+\sqrt{11}+\sqrt{23}+\sqrt{40}+\sqrt{59})$ - $(\sqrt{4}+\sqrt{22}+\sqrt{29}+\sqrt{30}+\sqrt{52})$
5	58	1.43×10^{-12}	$(\sqrt{5}+\sqrt{30}+\sqrt{37}+\sqrt{53}+\sqrt{58})$ - $(\sqrt{2}+\sqrt{34}+\sqrt{42}+\sqrt{55}+\sqrt{57})$
5	57	2.06×10^{-12}	$(\sqrt{26}+\sqrt{27}+\sqrt{29}+\sqrt{37}+\sqrt{46})$ - $(\sqrt{7}+\sqrt{30}+\sqrt{38}+\sqrt{45}+\sqrt{57})$
5	51...56	5.76×10^{-12}	$(\sqrt{12}+\sqrt{14}+\sqrt{37}+\sqrt{49}+\sqrt{50})$ - $(\sqrt{7}+\sqrt{22}+\sqrt{40}+\sqrt{43}+\sqrt{51})$
5	49, 50	2.25×10^{-11}	$(\sqrt{8}+\sqrt{10}+\sqrt{20}+\sqrt{47}+\sqrt{49})$ - $(\sqrt{15}+\sqrt{21}+\sqrt{26}+\sqrt{27}+\sqrt{31})$
5	33...48	2.53×10^{-11}	$(\sqrt{1}+\sqrt{18}+\sqrt{24}+\sqrt{30}+\sqrt{32})$ - $(\sqrt{6}+\sqrt{12}+\sqrt{15}+\sqrt{33}+\sqrt{33})$
5	31, 32	3.86×10^{-10}	$(\sqrt{6}+\sqrt{22}+\sqrt{22}+\sqrt{22}+\sqrt{22})$ - $(\sqrt{10}+\sqrt{11}+\sqrt{15}+\sqrt{28}+\sqrt{31})$
5	30	1.11×10^{-09}	$(\sqrt{6}+\sqrt{17}+\sqrt{17}+\sqrt{26}+\sqrt{30})$ - $(\sqrt{3}+\sqrt{19}+\sqrt{23}+\sqrt{25}+\sqrt{29})$
5	17...29	1.55×10^{-09}	$(\sqrt{3}+\sqrt{3}+\sqrt{14}+\sqrt{14}+\sqrt{17})$ - $(\sqrt{1}+\sqrt{10}+\sqrt{10}+\sqrt{15}+\sqrt{15})$
5	16	6.54×10^{-07}	$(\sqrt{3}+\sqrt{8}+\sqrt{11}+\sqrt{13}+\sqrt{16})$ - $(\sqrt{2}+\sqrt{6}+\sqrt{15}+\sqrt{15}+\sqrt{15})$

Continued on Next Page...

Table D.3 – Continued

k	n	r(n,k)	Sum of Square Roots
5	15	3.86×10^{-06}	$(\sqrt{3}+\sqrt{10}+\sqrt{11}+\sqrt{11}+\sqrt{14})$ - $(\sqrt{2}+\sqrt{5}+\sqrt{15}+\sqrt{15}+\sqrt{15})$
5	14	4.25×10^{-06}	$(\sqrt{2}+\sqrt{2}+\sqrt{11}+\sqrt{12}+\sqrt{13})$ - $(\sqrt{3}+\sqrt{6}+\sqrt{7}+\sqrt{7}+\sqrt{14})$
5	12, 13	4.82×10^{-06}	$(\sqrt{1}+\sqrt{1}+\sqrt{10}+\sqrt{12}+\sqrt{12})$ - $(\sqrt{2}+\sqrt{5}+\sqrt{6}+\sqrt{8}+\sqrt{10})$
5	10, 11	1.16×10^{-05}	$(\sqrt{3}+\sqrt{3}+\sqrt{6}+\sqrt{6}+\sqrt{6})$ - $(\sqrt{1}+\sqrt{2}+\sqrt{5}+\sqrt{9}+\sqrt{10})$
5	9	3.26×10^{-04}	$(\sqrt{5}+\sqrt{5}+\sqrt{5}+\sqrt{6}+\sqrt{7})$ - $(\sqrt{2}+\sqrt{3}+\sqrt{8}+\sqrt{8}+\sqrt{9})$
5	7, 8	9.56×10^{-04}	$(\sqrt{1}+\sqrt{5}+\sqrt{5}+\sqrt{7}+\sqrt{7})$ - $(\sqrt{2}+\sqrt{4}+\sqrt{6}+\sqrt{6}+\sqrt{6})$
5	6	1.98×10^{-03}	$(\sqrt{2}+\sqrt{3}+\sqrt{3}+\sqrt{5}+\sqrt{5})$ - $(\sqrt{1}+\sqrt{1}+\sqrt{6}+\sqrt{6}+\sqrt{6})$
5	5	6.57×10^{-03}	$(\sqrt{1}+\sqrt{1}+\sqrt{2}+\sqrt{2}+\sqrt{2})$ - $(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{5})$
5	4	2.53×10^{-02}	$(\sqrt{1}+\sqrt{1}+\sqrt{2}+\sqrt{4}+\sqrt{4})$ - $(\sqrt{2}+\sqrt{2}+\sqrt{2}+\sqrt{2}+\sqrt{3})$
5	3	9.64×10^{-02}	$(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{2}+\sqrt{2})$ - $(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{3})$
5	2	4.14×10^{-01}	$(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{2})$ - $(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1})$

Continued on Next Page...

Table D.3 – Continued

k	n	r(n,k)	Sum of Square Roots

Table D.4: The smallest difference of $k = 6$.

k	n	r(n,k)	Sum of Square Roots
6	149, 150	3.97×10^{-19}	$(\sqrt{34} + \sqrt{36} + \sqrt{57} + \sqrt{76} + \sqrt{92} + \sqrt{149})$ - $(\sqrt{11} + \sqrt{35} + \sqrt{52} + \sqrt{95} + \sqrt{139} + \sqrt{142})$
6	141...148	6.20×10^{-19}	$(\sqrt{9} + \sqrt{38} + \sqrt{42} + \sqrt{71} + \sqrt{118} + \sqrt{132})$ - $(\sqrt{6} + \sqrt{14} + \sqrt{82} + \sqrt{82} + \sqrt{105} + \sqrt{141})$
6	134...140	1.60×10^{-18}	$(\sqrt{4} + \sqrt{72} + \sqrt{75} + \sqrt{97} + \sqrt{129} + \sqrt{134})$ - $(\sqrt{28} + \sqrt{56} + \sqrt{69} + \sqrt{84} + \sqrt{103} + \sqrt{133})$
6	130...133	1.67×10^{-18}	$(\sqrt{18} + \sqrt{64} + \sqrt{86} + \sqrt{92} + \sqrt{101} + \sqrt{104})$ - $(\sqrt{28} + \sqrt{42} + \sqrt{51} + \sqrt{99} + \sqrt{123} + \sqrt{130})$
6	110...129	7.79×10^{-18}	$(\sqrt{12} + \sqrt{33} + \sqrt{49} + \sqrt{57} + \sqrt{79} + \sqrt{110})$ - $(\sqrt{15} + \sqrt{20} + \sqrt{70} + \sqrt{71} + \sqrt{78} + \sqrt{84})$
6	99...109	2.89×10^{-17}	$(\sqrt{21} + \sqrt{54} + \sqrt{62} + \sqrt{67} + \sqrt{92} + \sqrt{99})$ - $(\sqrt{15} + \sqrt{59} + \sqrt{76} + \sqrt{76} + \sqrt{82} + \sqrt{90})$
6	94...98	8.19×10^{-17}	$(\sqrt{6} + \sqrt{16} + \sqrt{22} + \sqrt{58} + \sqrt{75} + \sqrt{94})$ - $(\sqrt{2} + \sqrt{19} + \sqrt{28} + \sqrt{63} + \sqrt{80} + \sqrt{84})$

Continued on Next Page...

Table D.4 – Continued

k	n	r(n,k)	Sum of Square Roots
6	88...93	1.09×10^{-16}	$(\sqrt{9} + \sqrt{50} + \sqrt{59} + \sqrt{63} + \sqrt{71} + \sqrt{77})$ - $(\sqrt{22} + \sqrt{29} + \sqrt{51} + \sqrt{54} + \sqrt{80} + \sqrt{88})$
6	67...87	1.16×10^{-16}	$(\sqrt{7} + \sqrt{20} + \sqrt{29} + \sqrt{42} + \sqrt{52} + \sqrt{67})$ - $(\sqrt{16} + \sqrt{24} + \sqrt{27} + \sqrt{35} + \sqrt{39} + \sqrt{66})$
6	65, 66	4.54×10^{-16}	$(\sqrt{14} + \sqrt{15} + \sqrt{28} + \sqrt{42} + \sqrt{47} + \sqrt{65})$ - $(\sqrt{12} + \sqrt{17} + \sqrt{32} + \sqrt{45} + \sqrt{51} + \sqrt{52})$
6	59...64	7.56×10^{-15}	$(\sqrt{11} + \sqrt{15} + \sqrt{31} + \sqrt{32} + \sqrt{34} + \sqrt{40})$ - $(\sqrt{17} + \sqrt{17} + \sqrt{21} + \sqrt{21} + \sqrt{30} + \sqrt{59})$
6	57, 58	7.41×10^{-14}	$(\sqrt{17} + \sqrt{23} + \sqrt{32} + \sqrt{42} + \sqrt{55} + \sqrt{57})$ - $(\sqrt{19} + \sqrt{24} + \sqrt{39} + \sqrt{41} + \sqrt{44} + \sqrt{56})$
6	54...56	4.00×10^{-13}	$(\sqrt{10} + \sqrt{23} + \sqrt{32} + \sqrt{38} + \sqrt{43} + \sqrt{54})$ - $(\sqrt{13} + \sqrt{24} + \sqrt{30} + \sqrt{31} + \sqrt{47} + \sqrt{53})$
6	53	4.00×10^{-13}	$(\sqrt{6} + \sqrt{23} + \sqrt{32} + \sqrt{38} + \sqrt{40} + \sqrt{43})$ - $(\sqrt{10} + \sqrt{13} + \sqrt{30} + \sqrt{31} + \sqrt{47} + \sqrt{53})$
6	50...52	4.13×10^{-13}	$(\sqrt{10} + \sqrt{19} + \sqrt{20} + \sqrt{38} + \sqrt{42} + \sqrt{50})$ - $(\sqrt{17} + \sqrt{25} + \sqrt{26} + \sqrt{26} + \sqrt{34} + \sqrt{43})$
6	40...49	1.20×10^{-12}	$(\sqrt{5} + \sqrt{9} + \sqrt{22} + \sqrt{26} + \sqrt{31} + \sqrt{39})$ - $(\sqrt{7} + \sqrt{10} + \sqrt{17} + \sqrt{21} + \sqrt{36} + \sqrt{40})$
6	36...39	2.53×10^{-11}	$(\sqrt{6} + \sqrt{18} + \sqrt{26} + \sqrt{30} + \sqrt{32} + \sqrt{36})$ - $(\sqrt{12} + \sqrt{15} + \sqrt{25} + \sqrt{26} + \sqrt{33} + \sqrt{33})$
6	31...35	7.76×10^{-11}	$(\sqrt{11} + \sqrt{13} + \sqrt{24} + \sqrt{25} + \sqrt{27} + \sqrt{31})$ - $(\sqrt{14} + \sqrt{15} + \sqrt{22} + \sqrt{23} + \sqrt{26} + \sqrt{29})$

Continued on Next Page...

Table D.4 – Continued

k	n	r(n,k)	Sum of Square Roots
6	26...30	4.91×10^{-10}	$(\sqrt{8}+\sqrt{9}+\sqrt{14}+\sqrt{20}+\sqrt{21}+\sqrt{26})$ - $(\sqrt{15}+\sqrt{15}+\sqrt{15}+\sqrt{15}+\sqrt{15}+\sqrt{19})$
6	24, 25	1.06×10^{-09}	$(\sqrt{7}+\sqrt{9}+\sqrt{11}+\sqrt{15}+\sqrt{17}+\sqrt{19})$ - $(\sqrt{2}+\sqrt{5}+\sqrt{13}+\sqrt{20}+\sqrt{22}+\sqrt{24})$
6	17...23	1.55×10^{-09}	$(\sqrt{3}+\sqrt{3}+\sqrt{8}+\sqrt{14}+\sqrt{14}+\sqrt{17})$ - $(\sqrt{1}+\sqrt{8}+\sqrt{10}+\sqrt{10}+\sqrt{15}+\sqrt{15})$
6	15, 16	5.19×10^{-07}	$(\sqrt{6}+\sqrt{8}+\sqrt{10}+\sqrt{12}+\sqrt{12}+\sqrt{14})$ - $(\sqrt{2}+\sqrt{11}+\sqrt{11}+\sqrt{11}+\sqrt{15}+\sqrt{15})$
6	13, 14	9.90×10^{-07}	$(\sqrt{2}+\sqrt{8}+\sqrt{13}+\sqrt{13}+\sqrt{13}+\sqrt{13})$ - $(\sqrt{5}+\sqrt{10}+\sqrt{11}+\sqrt{11}+\sqrt{11}+\sqrt{11})$
6	12	4.82×10^{-06}	$(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{10}+\sqrt{12}+\sqrt{12})$ - $(\sqrt{1}+\sqrt{2}+\sqrt{5}+\sqrt{6}+\sqrt{8}+\sqrt{10})$
6	7...11	4.82×10^{-06}	$(\sqrt{3}+\sqrt{3}+\sqrt{3}+\sqrt{3}+\sqrt{4}+\sqrt{7})$ - $(\sqrt{2}+\sqrt{2}+\sqrt{2}+\sqrt{5}+\sqrt{6}+\sqrt{7})$
6	6	4.82×10^{-06}	$(\sqrt{3}+\sqrt{3}+\sqrt{3}+\sqrt{3}+\sqrt{4}+\sqrt{5})$ - $(\sqrt{2}+\sqrt{2}+\sqrt{2}+\sqrt{5}+\sqrt{5}+\sqrt{6})$
6	5	6.57×10^{-03}	$(\sqrt{2}+\sqrt{2}+\sqrt{2}+\sqrt{2}+\sqrt{2}+\sqrt{3})$ - $(\sqrt{1}+\sqrt{1}+\sqrt{2}+\sqrt{2}+\sqrt{3}+\sqrt{5})$
6	4	7.52×10^{-02}	$(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{2}+\sqrt{3}+\sqrt{4})$ - $(\sqrt{1}+\sqrt{2}+\sqrt{2}+\sqrt{2}+\sqrt{2}+\sqrt{2})$
6	3	9.64×10^{-02}	$(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{2}+\sqrt{2})$ - $(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{3})$

Continued on Next Page...

Table D.4 – Continued

k	n	r(n,k)	Sum of Square Roots
6	2	4.14×10^{-01}	$(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{2})$ - $(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1})$

Table D.5: The smallest difference of $k = 7$.

k	n	r(n,k)	Sum of Square Roots
7	85...100	1.88×10^{-19}	$(\sqrt{7}+\sqrt{14}+\sqrt{39}+\sqrt{70}+\sqrt{72}+\sqrt{76}+\sqrt{85})$ - $(\sqrt{13}+\sqrt{16}+\sqrt{46}+\sqrt{55}+\sqrt{67}+\sqrt{73}+\sqrt{79})$
7	80...84	1.88×10^{-18}	$(\sqrt{19}+\sqrt{29}+\sqrt{42}+\sqrt{42}+\sqrt{51}+\sqrt{62}+\sqrt{70})$ - $(\sqrt{4}+\sqrt{5}+\sqrt{53}+\sqrt{69}+\sqrt{75}+\sqrt{75}+\sqrt{80})$
7	72...79	5.44×10^{-18}	$(\sqrt{21}+\sqrt{22}+\sqrt{34}+\sqrt{45}+\sqrt{49}+\sqrt{58}+\sqrt{61})$ - $(\sqrt{7}+\sqrt{15}+\sqrt{40}+\sqrt{42}+\sqrt{65}+\sqrt{70}+\sqrt{72})$
7	71	3.83×10^{-17}	$(\sqrt{1}+\sqrt{10}+\sqrt{41}+\sqrt{51}+\sqrt{51}+\sqrt{61}+\sqrt{71})$ - $(\sqrt{8}+\sqrt{20}+\sqrt{28}+\sqrt{38}+\sqrt{45}+\sqrt{60}+\sqrt{62})$
7	62...70	6.46×10^{-17}	$(\sqrt{5}+\sqrt{15}+\sqrt{23}+\sqrt{32}+\sqrt{37}+\sqrt{55}+\sqrt{62})$ - $(\sqrt{4}+\sqrt{19}+\sqrt{26}+\sqrt{31}+\sqrt{33}+\sqrt{56}+\sqrt{59})$
7	61	2.37×10^{-16}	$(\sqrt{10}+\sqrt{35}+\sqrt{36}+\sqrt{43}+\sqrt{43}+\sqrt{52}+\sqrt{59})$ - $(\sqrt{6}+\sqrt{30}+\sqrt{46}+\sqrt{46}+\sqrt{47}+\sqrt{48}+\sqrt{61})$

Continued on Next Page...

Table D.5 – Continued

k	n	r(n,k)	Sum of Square Roots
7	54...60	2.97×10^{-16}	$(\sqrt{21} + \sqrt{31} + \sqrt{34} + \sqrt{36} + \sqrt{41} + \sqrt{51} + \sqrt{54})$ - $(\sqrt{28} + \sqrt{32} + \sqrt{35} + \sqrt{37} + \sqrt{37} + \sqrt{44} + \sqrt{52})$
7	51...53	4.57×10^{-16}	$(\sqrt{13} + \sqrt{17} + \sqrt{26} + \sqrt{36} + \sqrt{38} + \sqrt{39} + \sqrt{46})$ - $(\sqrt{11} + \sqrt{14} + \sqrt{15} + \sqrt{40} + \sqrt{42} + \sqrt{51} + \sqrt{51})$
7	41...50	3.43×10^{-15}	$(\sqrt{6} + \sqrt{11} + \sqrt{21} + \sqrt{26} + \sqrt{35} + \sqrt{38} + \sqrt{39})$ - $(\sqrt{7} + \sqrt{10} + \sqrt{19} + \sqrt{23} + \sqrt{37} + \sqrt{40} + \sqrt{41})$
7	34...40	2.80×10^{-13}	$(\sqrt{11} + \sqrt{16} + \sqrt{21} + \sqrt{23} + \sqrt{26} + \sqrt{29} + \sqrt{29})$ - $(\sqrt{10} + \sqrt{13} + \sqrt{22} + \sqrt{24} + \sqrt{24} + \sqrt{30} + \sqrt{34})$
7	33	1.15×10^{-12}	$(\sqrt{11} + \sqrt{12} + \sqrt{17} + \sqrt{17} + \sqrt{19} + \sqrt{26} + \sqrt{30})$ - $(\sqrt{7} + \sqrt{13} + \sqrt{14} + \sqrt{18} + \sqrt{22} + \sqrt{28} + \sqrt{33})$
7	29...32	1.02×10^{-11}	$(\sqrt{8} + \sqrt{9} + \sqrt{10} + \sqrt{19} + \sqrt{19} + \sqrt{20} + \sqrt{21})$ - $(\sqrt{5} + \sqrt{5} + \sqrt{13} + \sqrt{13} + \sqrt{23} + \sqrt{24} + \sqrt{29})$
7	28	9.66×10^{-11}	$(\sqrt{2} + \sqrt{2} + \sqrt{11} + \sqrt{14} + \sqrt{26} + \sqrt{26} + \sqrt{28})$ - $(\sqrt{1} + \sqrt{13} + \sqrt{13} + \sqrt{17} + \sqrt{17} + \sqrt{17} + \sqrt{23})$
7	22...27	2.42×10^{-10}	$(\sqrt{3} + \sqrt{10} + \sqrt{11} + \sqrt{11} + \sqrt{17} + \sqrt{21} + \sqrt{22})$ - $(\sqrt{6} + \sqrt{7} + \sqrt{14} + \sqrt{14} + \sqrt{15} + \sqrt{16} + \sqrt{20})$
7	21	1.55×10^{-09}	$(\sqrt{3} + \sqrt{3} + \sqrt{3} + \sqrt{14} + \sqrt{14} + \sqrt{17} + \sqrt{21})$ - $(\sqrt{1} + \sqrt{3} + \sqrt{10} + \sqrt{10} + \sqrt{15} + \sqrt{15} + \sqrt{21})$
7	19, 20	1.55×10^{-09}	$(\sqrt{3} + \sqrt{3} + \sqrt{3} + \sqrt{14} + \sqrt{14} + \sqrt{17} + \sqrt{19})$ - $(\sqrt{1} + \sqrt{3} + \sqrt{10} + \sqrt{10} + \sqrt{15} + \sqrt{15} + \sqrt{19})$
7	17, 18	1.55×10^{-09}	$(\sqrt{3} + \sqrt{3} + \sqrt{3} + \sqrt{14} + \sqrt{14} + \sqrt{17} + \sqrt{17})$ - $(\sqrt{1} + \sqrt{3} + \sqrt{10} + \sqrt{10} + \sqrt{15} + \sqrt{15} + \sqrt{17})$

Continued on Next Page...

Table D.5 – Continued

k	n	r(n,k)	Sum of Square Roots
7	14...16	4.96×10^{-08}	$(\sqrt{7}+\sqrt{8}+\sqrt{8}+\sqrt{11}+\sqrt{13}+\sqrt{13}+\sqrt{14})$ - $(\sqrt{2}+\sqrt{6}+\sqrt{14}+\sqrt{14}+\sqrt{14}+\sqrt{14}+\sqrt{14})$
7	11...13	1.49×10^{-07}	$(\sqrt{5}+\sqrt{5}+\sqrt{5}+\sqrt{5}+\sqrt{5}+\sqrt{5}+\sqrt{10})$ - $(\sqrt{3}+\sqrt{3}+\sqrt{6}+\sqrt{6}+\sqrt{6}+\sqrt{6}+\sqrt{11})$
7	10	4.82×10^{-06}	$(\sqrt{3}+\sqrt{3}+\sqrt{3}+\sqrt{3}+\sqrt{4}+\sqrt{8}+\sqrt{10})$ - $(\sqrt{2}+\sqrt{2}+\sqrt{2}+\sqrt{5}+\sqrt{6}+\sqrt{8}+\sqrt{10})$
7	7...9	4.82×10^{-06}	$(\sqrt{3}+\sqrt{3}+\sqrt{3}+\sqrt{3}+\sqrt{4}+\sqrt{7}+\sqrt{7})$ - $(\sqrt{2}+\sqrt{2}+\sqrt{2}+\sqrt{5}+\sqrt{6}+\sqrt{7}+\sqrt{7})$
7	6	4.82×10^{-06}	$(\sqrt{2}+\sqrt{3}+\sqrt{3}+\sqrt{3}+\sqrt{3}+\sqrt{4}+\sqrt{5})$ - $(\sqrt{2}+\sqrt{2}+\sqrt{2}+\sqrt{2}+\sqrt{5}+\sqrt{5}+\sqrt{6})$
7	5	6.57×10^{-03}	$(\sqrt{1}+\sqrt{1}+\sqrt{2}+\sqrt{2}+\sqrt{2}+\sqrt{2}+\sqrt{3})$ - $(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{2}+\sqrt{3}+\sqrt{5})$
7	4	2.53×10^{-02}	$(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{4}+\sqrt{4})$ - $(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{2}+\sqrt{2}+\sqrt{2}+\sqrt{3})$
7	3	9.64×10^{-02}	$(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{2}+\sqrt{2})$ - $(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{3})$
7	2	4.14×10^{-01}	$(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{2})$ - $(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1})$

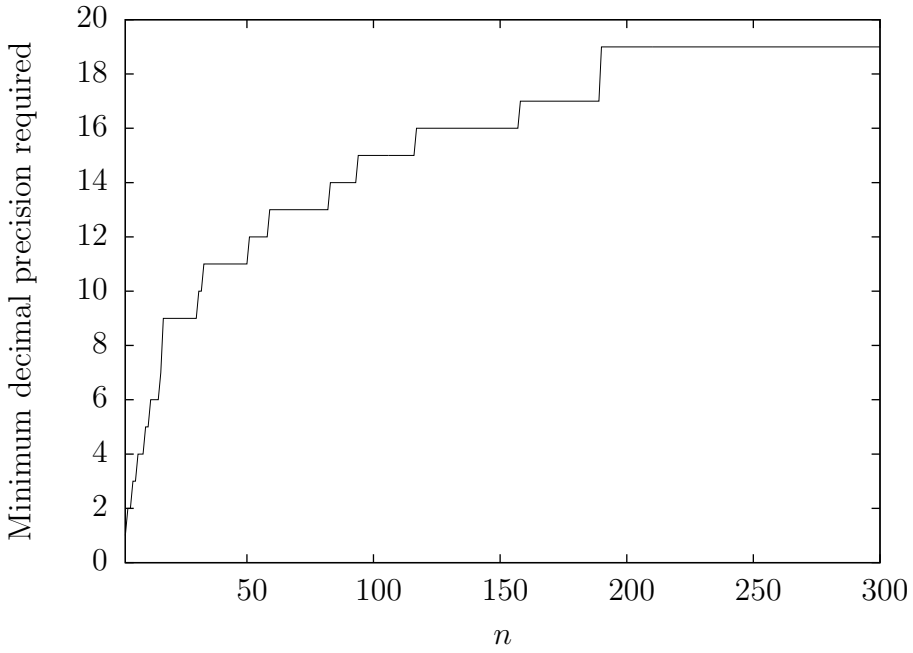

Figure D.3: Minimum precision required when n varies and $k = 5$.

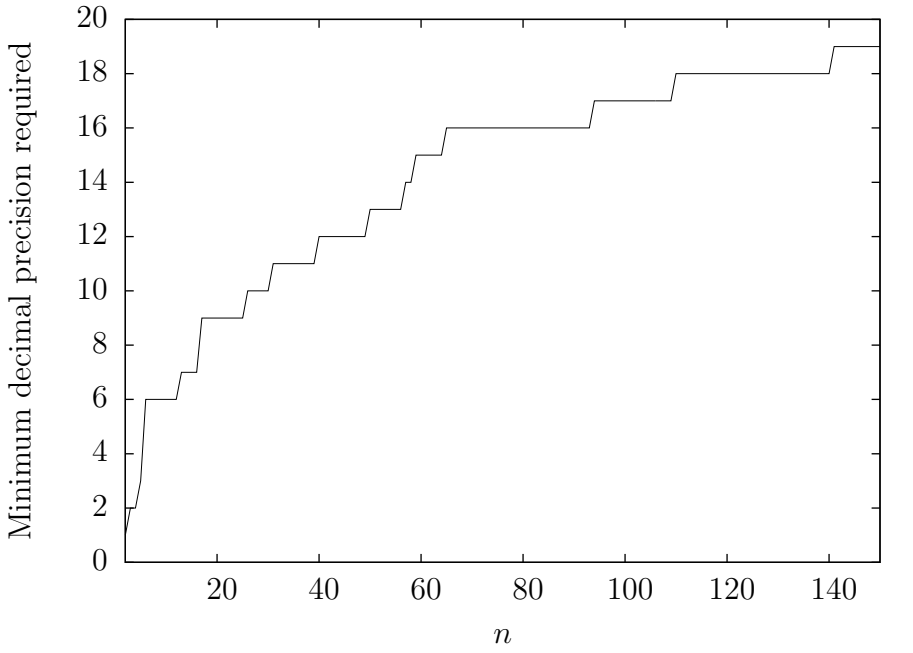

Figure D.4: Minimum precision required when n varies and $k = 6$.

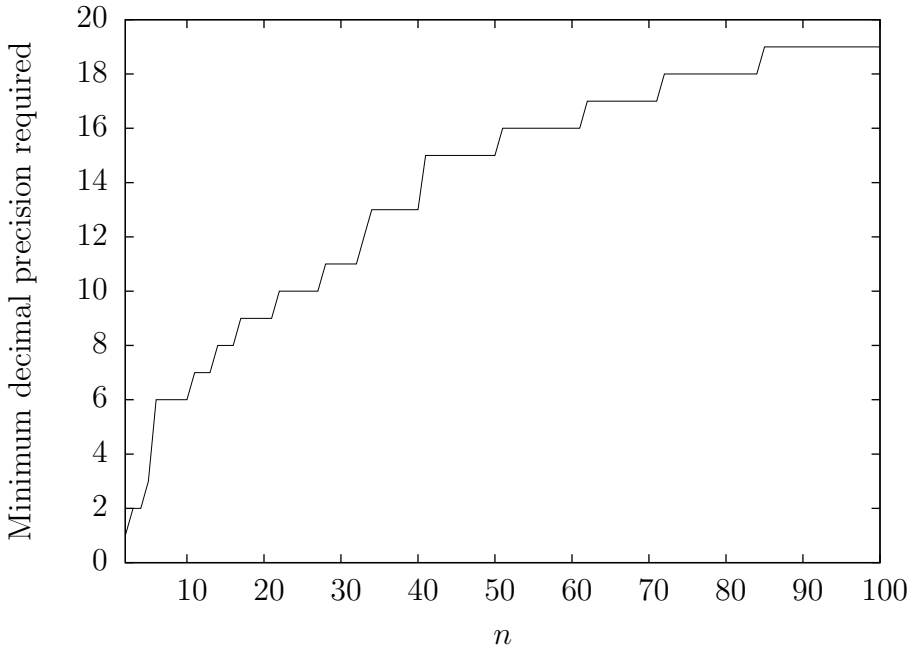

Figure D.5: Minimum precision required when n varies and $k = 7$.

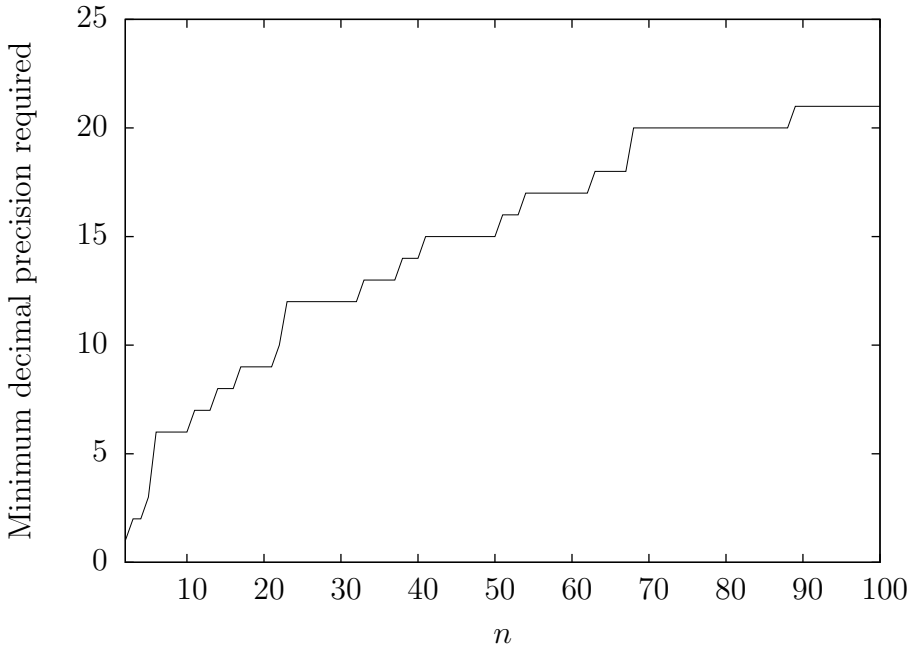

Figure D.6: Minimum precision required when n varies and $k = 8$.

Table D.6: The smallest difference of $k = 8$.

k	n	r(n,k)	Sum of Square Roots
8	97...100	2.77×10^{-21}	$(\sqrt{16} + \sqrt{43} + \sqrt{43} + \sqrt{46} + \sqrt{60} + \sqrt{85} + \sqrt{89} + \sqrt{95})$ - $(\sqrt{7} + \sqrt{41} + \sqrt{42} + \sqrt{51} + \sqrt{76} + \sqrt{83} + \sqrt{94} + \sqrt{97})$
8	89...96	4.53×10^{-21}	$(\sqrt{13} + \sqrt{21} + \sqrt{28} + \sqrt{34} + \sqrt{47} + \sqrt{68} + \sqrt{84} + \sqrt{89})$ - $(\sqrt{11} + \sqrt{30} + \sqrt{33} + \sqrt{46} + \sqrt{53} + \sqrt{54} + \sqrt{65} + \sqrt{81})$
8	86...88	3.20×10^{-20}	$(\sqrt{28} + \sqrt{39} + \sqrt{42} + \sqrt{42} + \sqrt{44} + \sqrt{60} + \sqrt{60} + \sqrt{84})$ - $(\sqrt{10} + \sqrt{22} + \sqrt{36} + \sqrt{48} + \sqrt{67} + \sqrt{75} + \sqrt{79} + \sqrt{86})$
8	68...85	4.48×10^{-20}	$(\sqrt{13} + \sqrt{27} + \sqrt{46} + \sqrt{51} + \sqrt{57} + \sqrt{59} + \sqrt{60} + \sqrt{65})$ - $(\sqrt{24} + \sqrt{35} + \sqrt{37} + \sqrt{38} + \sqrt{45} + \sqrt{62} + \sqrt{62} + \sqrt{68})$
8	67	2.22×10^{-18}	$(\sqrt{8} + \sqrt{27} + \sqrt{28} + \sqrt{47} + \sqrt{53} + \sqrt{55} + \sqrt{61} + \sqrt{64})$ - $(\sqrt{2} + \sqrt{10} + \sqrt{39} + \sqrt{57} + \sqrt{62} + \sqrt{65} + \sqrt{67} + \sqrt{67})$
8	66	3.61×10^{-18}	$(\sqrt{14} + \sqrt{21} + \sqrt{27} + \sqrt{34} + \sqrt{36} + \sqrt{47} + \sqrt{52} + \sqrt{66})$ - $(\sqrt{11} + \sqrt{15} + \sqrt{23} + \sqrt{38} + \sqrt{38} + \sqrt{57} + \sqrt{58} + \sqrt{65})$
8	63...65	9.56×10^{-18}	$(\sqrt{1} + \sqrt{20} + \sqrt{30} + \sqrt{32} + \sqrt{39} + \sqrt{40} + \sqrt{53} + \sqrt{58})$ - $(\sqrt{6} + \sqrt{14} + \sqrt{16} + \sqrt{26} + \sqrt{33} + \sqrt{57} + \sqrt{57} + \sqrt{63})$
8	54...62	1.15×10^{-17}	$(\sqrt{12} + \sqrt{18} + \sqrt{25} + \sqrt{29} + \sqrt{41} + \sqrt{41} + \sqrt{42} + \sqrt{54})$ - $(\sqrt{13} + \sqrt{22} + \sqrt{23} + \sqrt{34} + \sqrt{37} + \sqrt{38} + \sqrt{43} + \sqrt{49})$
8	51...53	4.57×10^{-16}	$(\sqrt{11} + \sqrt{13} + \sqrt{17} + \sqrt{26} + \sqrt{38} + \sqrt{39} + \sqrt{46} + \sqrt{49})$ - $(\sqrt{1} + \sqrt{14} + \sqrt{15} + \sqrt{40} + \sqrt{42} + \sqrt{44} + \sqrt{51} + \sqrt{51})$
8	46...50	1.12×10^{-15}	$(\sqrt{20} + \sqrt{22} + \sqrt{25} + \sqrt{35} + \sqrt{35} + \sqrt{42} + \sqrt{43} + \sqrt{46})$ - $(\sqrt{17} + \sqrt{29} + \sqrt{32} + \sqrt{33} + \sqrt{33} + \sqrt{33} + \sqrt{45} + \sqrt{45})$
8	44, 45	2.32×10^{-15}	$(\sqrt{11} + \sqrt{11} + \sqrt{17} + \sqrt{17} + \sqrt{22} + \sqrt{41} + \sqrt{43} + \sqrt{44})$

Continued on Next Page...

Table D.6 – Continued

k	n	r(n,k)	Sum of Square Roots
			- $(\sqrt{10}+\sqrt{10}+\sqrt{23}+\sqrt{26}+\sqrt{26}+\sqrt{27}+\sqrt{40}+\sqrt{40})$
8	41...43	3.43×10^{-15}	$(\sqrt{6}+\sqrt{7}+\sqrt{11}+\sqrt{21}+\sqrt{26}+\sqrt{35}+\sqrt{38}+\sqrt{39})$ - $(\sqrt{10}+\sqrt{10}+\sqrt{10}+\sqrt{19}+\sqrt{23}+\sqrt{28}+\sqrt{37}+\sqrt{41})$
8	38...40	1.45×10^{-14}	$(\sqrt{3}+\sqrt{8}+\sqrt{10}+\sqrt{28}+\sqrt{31}+\sqrt{32}+\sqrt{33}+\sqrt{35})$ - $(\sqrt{10}+\sqrt{13}+\sqrt{14}+\sqrt{19}+\sqrt{21}+\sqrt{24}+\sqrt{29}+\sqrt{38})$
8	37	2.01×10^{-13}	$(\sqrt{3}+\sqrt{15}+\sqrt{22}+\sqrt{26}+\sqrt{26}+\sqrt{33}+\sqrt{33}+\sqrt{36})$ - $(\sqrt{6}+\sqrt{13}+\sqrt{17}+\sqrt{21}+\sqrt{30}+\sqrt{34}+\sqrt{34}+\sqrt{37})$
8	33...36	2.57×10^{-13}	$(\sqrt{10}+\sqrt{11}+\sqrt{11}+\sqrt{19}+\sqrt{22}+\sqrt{23}+\sqrt{24}+\sqrt{24})$ - $(\sqrt{5}+\sqrt{6}+\sqrt{9}+\sqrt{15}+\sqrt{25}+\sqrt{31}+\sqrt{31}+\sqrt{33})$
8	31, 32	3.61×10^{-12}	$(\sqrt{7}+\sqrt{10}+\sqrt{14}+\sqrt{15}+\sqrt{21}+\sqrt{22}+\sqrt{23}+\sqrt{31})$ - $(\sqrt{5}+\sqrt{8}+\sqrt{11}+\sqrt{16}+\sqrt{24}+\sqrt{27}+\sqrt{28}+\sqrt{28})$
8	24...30	4.78×10^{-12}	$(\sqrt{2}+\sqrt{3}+\sqrt{3}+\sqrt{11}+\sqrt{15}+\sqrt{21}+\sqrt{21}+\sqrt{24})$ - $(\sqrt{1}+\sqrt{7}+\sqrt{8}+\sqrt{10}+\sqrt{10}+\sqrt{14}+\sqrt{23}+\sqrt{23})$
8	23	4.78×10^{-12}	$(\sqrt{3}+\sqrt{3}+\sqrt{6}+\sqrt{6}+\sqrt{11}+\sqrt{15}+\sqrt{21}+\sqrt{21})$ - $(\sqrt{1}+\sqrt{2}+\sqrt{7}+\sqrt{10}+\sqrt{10}+\sqrt{14}+\sqrt{23}+\sqrt{23})$
8	22	6.82×10^{-10}	$(\sqrt{1}+\sqrt{1}+\sqrt{5}+\sqrt{14}+\sqrt{15}+\sqrt{15}+\sqrt{22}+\sqrt{22})$ - $(\sqrt{6}+\sqrt{6}+\sqrt{6}+\sqrt{7}+\sqrt{11}+\sqrt{13}+\sqrt{13}+\sqrt{21})$
8	21	1.55×10^{-09}	$(\sqrt{2}+\sqrt{3}+\sqrt{3}+\sqrt{14}+\sqrt{14}+\sqrt{17}+\sqrt{19}+\sqrt{21})$ - $(\sqrt{1}+\sqrt{2}+\sqrt{10}+\sqrt{10}+\sqrt{15}+\sqrt{15}+\sqrt{19}+\sqrt{21})$
8	19, 20	1.55×10^{-09}	$(\sqrt{2}+\sqrt{3}+\sqrt{3}+\sqrt{14}+\sqrt{14}+\sqrt{17}+\sqrt{19}+\sqrt{19})$ - $(\sqrt{1}+\sqrt{2}+\sqrt{10}+\sqrt{10}+\sqrt{15}+\sqrt{15}+\sqrt{19}+\sqrt{19})$
8	17, 18	1.55×10^{-09}	$(\sqrt{2}+\sqrt{3}+\sqrt{3}+\sqrt{14}+\sqrt{14}+\sqrt{17}+\sqrt{17}+\sqrt{17})$

Continued on Next Page...

Table D.6 – Continued

k	n	r(n,k)	Sum of Square Roots
			- $(\sqrt{1}+\sqrt{2}+\sqrt{10}+\sqrt{10}+\sqrt{15}+\sqrt{15}+\sqrt{17}+\sqrt{17})$
8	15, 16	4.96×10^{-08}	$(\sqrt{7}+\sqrt{8}+\sqrt{8}+\sqrt{11}+\sqrt{13}+\sqrt{13}+\sqrt{14}+\sqrt{15})$ - $(\sqrt{2}+\sqrt{6}+\sqrt{14}+\sqrt{14}+\sqrt{14}+\sqrt{14}+\sqrt{14}+\sqrt{15})$
8	14	4.96×10^{-08}	$(\sqrt{7}+\sqrt{8}+\sqrt{8}+\sqrt{10}+\sqrt{11}+\sqrt{13}+\sqrt{13}+\sqrt{14})$ - $(\sqrt{2}+\sqrt{6}+\sqrt{10}+\sqrt{14}+\sqrt{14}+\sqrt{14}+\sqrt{14}+\sqrt{14})$
8	11...13	1.49×10^{-07}	$(\sqrt{1}+\sqrt{5}+\sqrt{5}+\sqrt{5}+\sqrt{5}+\sqrt{5}+\sqrt{5}+\sqrt{10})$ - $(\sqrt{1}+\sqrt{3}+\sqrt{3}+\sqrt{6}+\sqrt{6}+\sqrt{6}+\sqrt{6}+\sqrt{11})$
8	10	4.82×10^{-06}	$(\sqrt{3}+\sqrt{3}+\sqrt{3}+\sqrt{3}+\sqrt{3}+\sqrt{5}+\sqrt{9}+\sqrt{10})$ - $(\sqrt{1}+\sqrt{2}+\sqrt{3}+\sqrt{5}+\sqrt{5}+\sqrt{6}+\sqrt{8}+\sqrt{10})$
8	9	4.82×10^{-06}	$(\sqrt{3}+\sqrt{3}+\sqrt{3}+\sqrt{3}+\sqrt{3}+\sqrt{5}+\sqrt{7}+\sqrt{9})$ - $(\sqrt{1}+\sqrt{2}+\sqrt{3}+\sqrt{5}+\sqrt{5}+\sqrt{6}+\sqrt{7}+\sqrt{8})$
8	6...8	4.82×10^{-06}	$(\sqrt{3}+\sqrt{3}+\sqrt{3}+\sqrt{3}+\sqrt{4}+\sqrt{6}+\sqrt{6}+\sqrt{6})$ - $(\sqrt{2}+\sqrt{2}+\sqrt{2}+\sqrt{5}+\sqrt{6}+\sqrt{6}+\sqrt{6}+\sqrt{6})$
8	5	1.46×10^{-03}	$(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{2}+\sqrt{2}+\sqrt{5}+\sqrt{5}+\sqrt{5})$ - $(\sqrt{2}+\sqrt{2}+\sqrt{2}+\sqrt{2}+\sqrt{2}+\sqrt{3}+\sqrt{3}+\sqrt{4})$
8	4	2.53×10^{-02}	$(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{4}+\sqrt{4})$ - $(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{2}+\sqrt{2}+\sqrt{2}+\sqrt{3})$
8	3	9.64×10^{-02}	$(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{2}+\sqrt{2})$ - $(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{3})$
8	2	4.14×10^{-01}	$(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{2})$ - $(\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1}+\sqrt{1})$